AFRL-IF-WP-TR-2004-1517

SECURITY/TRUST AS A POLYMORPHIC COMPUTING CONSTRAINT

Clark Weissman, Brant Hashii, and Jerry Cole

Northrop Grumman Corporation
Integrated Systems, MS 9L74/W6
One Hornet Way
El Segundo, CA 90245-2804

SEPTEMBER 2003

Final Report for 14 May 2001 – 30 September 2003

STINFO FINAL REPORT

INFORMATION DIRECTORATE
AIR FORCE RESEARCH LABORATORY
AIR FORCE MATERIEL COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OH 45433-7334

Kerry L Hill

KERRY L. HILL
Project Engineer
Embedded Info Sys Engineering Branch
Information Systems Technology Division

Alfred J. Scarpelli

ALFRED J. SCARPELLI
Team Leader
Embedded Info Systems Engineering Branch
Information Systems Technology Division

[illegible]

JAMES S. WILLIAMSON, Chief
Embedded Info Systems Engineering Branch
Information Systems Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. REPORT DATE *(DD-MM-YY)* September 2003	**2. REPORT TYPE** Final	**3. DATES COVERED *(From - To)*** 05/14/2001 – 09/30/2003

4. TITLE AND SUBTITLE SECURITY/TRUST AS A POLYMORPHIC COMPUTING CONSTRAINT	**5a. CONTRACT NUMBER** F33615-01-C-1891
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER 62712E
6. AUTHOR(S) Clark Weissman, Brant Hashii, and Jerry Cole	**5d. PROJECT NUMBER** L454
	5e. TASK NUMBER 18
	5f. WORK UNIT NUMBER 91
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Northrop Grumman Corporation Integrated Systems, MS 9L74/W6 One Hornet Way El Segundo, CA 90245-2804	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Information Directorate Air Force Research Laboratory Air Force Materiel Command Wright-Patterson AFB, OH 45433-7334 DARPA/IPTO 3701 Fairfax Drive Arlington, VA 22203-1714	**10. SPONSORING/MONITORING AGENCY ACRONYM(S)** AFRL/IFTA **11. SPONSORING/MONITORING AGENCY REPORT NUMBER(S)** AFRL-IF-WP-TR-2004-1517

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

DoD Joint Vision 2020 (JV2020) is the integrated multi-service planning document for conduct among coalition forces of future warfare. It requires the confluence of a number of key avionics technical developments: integrating the network-centric battlefield, management of hundred thousands of distributed processors, high assurance Multi Level Security (MLS) in the battlefield, and low cost high assurance engineering. The paper describes the results of a study and modeling of a new security architecture (MLS-PCA) that yields a practical solution for JV2020 based upon DARPA Polymorphic Computing Architecture (PCA) advances and a new distributed process-level encryption scheme. The paper defines a functional model and a verified formal specification of MLS-PCA, for high assurance, with the constraints PCA software, hardware, and morphware must support. Also, the paper shows a viable mapping of the MLS-PCA model to the PCA hardware. MLS-PCA is designed to support upwards of 400,000 CPUs predicted by Moore's law to be available circa 2020.

15. SUBJECT TERMS

Multi Level Security (MLS), Polymorphic Computer Architecture (PCA), High assurance network security via process-level encryption, DoD Joint Vision 2020 (JV2020), Formal modeling and formal specification, Trusted avionics architecture for hundreds of thousands of processors predicted by 2020 by Moore's Law

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT:**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** (Monitor)
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Kerry Hill
Unclassified	Unclassified	Unclassified	SAR	56	**19b. TELEPHONE NUMBER *(Include Area Code)*** (937) 255-6548 x3604

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTIONS AND MOTIVATION

DOD Joint Vision 2020 describes the future battle space consisting of space, air, land, sea, and undersea forces integrated via a global network of sensors, command and control, communications, and integrated strike warfare elements [JV2020]. The Achilles heel of this network-centric vision is the high assurance Multi Level Security (MLS) that permits the myriad communications that make JV2020 possible. MLS research and development over the past two decades has defined the requirements that must be satisfied for DOD systems [Anderson, TCSEC, CC99, Rainbow]. However, the high cost of developing and certifying high assurance systems to these requirements has been prohibitive and development time has been excessively long. Innovative use of Polymorphous Computing Architecture (PCA) to satisfy these MLS requirements in a scheme at process-level granularity is a novel R&D approach that simplifies system design, yet provides flexible configurable MLS systems. Such systems can meet security requirements to support different secure data streams in battlefield network-centric computing, as advocated in Joint Vision 2020. Many additional security requirements can be satisfied concurrently, including message integrity, authentication, confidentiality, code mobility, and dynamic coalitions. This paper describes a new security architecture to employ the richness of processing logic expected by 2020, such as the DARPA Polymorphic Computing Architecture (PCA) program [PCA].

The PCA goals are to span a broad dynamic application space by implementing a transparent reactive layer between an embedded avionics application program and the malleable micro-architecture elements on which it will operate. This polymorphic layer will enable software and hardware to be developed in a cooperative constraint sensitive environment instead of in a failure prone hardware first and software last paradigm. The PCA program will implement a family of novel malleable micro-architecture processing elements, i.e., PCA chips, to include compute cores, caches, memory structures, data paths, network interfaces, network fabrics with incremental instructions, OS, and network protocols. These elements will have the ability to reconfigure to match changing mission and scenario demands. To support the use of polymorphous computing systems, the program will create a model based software framework for reactive monitoring, optimization, modeling, resource negotiation and allocation, regeneration, and verification.

Our new security architecture, MLS-PCA, is the focus of this report, which covers the novel functional architecture, a formal specification model, and an examination of the security constraints that must be imposed on the PCA software-morphing layer and on the underlying chip hardware. We determined a set of security constraints that must be imposed on the PCA hardware to allow PCA to achieve certifiably secure MLS avionics processing. These are not PCA constraints, or deficiencies in the polymorphous computing architecture, or limitations on the morphware compiler tools. They are security constraints to restrict the PCA flexibility sufficiently to permit security certification of the resulting MLS-PCA-based avionics systems.

2 BACKGROUND

2.1 Character of Avionics

Legacy military avionics systems were developed using a "federated architecture" in which each subsystem was logically and physically separate. Each had its own set of component parts, which could not be used to support other subsystems in times of equipment failure. This was the approach taken with the F-15, F-16 and F/A-18, in the 1970s. In the early 1980s, the Department of Defense put together the "Pave Pillar" architecture that led to the Joint Integrated Avionics Working Group (JIAWG) Advanced Avionics Architecture. The result of this integrated avionics architecture was that computational resources could be interconnected by high speed networks to allow for more flexible usage of these resources, e.g., re-assigning a processor to take over the function of a failed processor. PCA is the next step in Pave Pillar integrated avionics architecture with the multiple processors, memory, and connecting busses all contained on a single chip. This also led to the ability to share information, e.g., to utilize fusion methods to merge radar and electro-optical information to create an improved way to convey information to the pilot. The pilot no longer had to mentally perform the integration function from a variety of gauges and instruments. Unfortunately, the sharing of information resources in a classified avionics environment leads to another challenge; either 1) operate at "System High", with a labor intensive burden of separating out the different classification levels at the end of a mission, or 2) solve the MLS problem. The combination of highly classified data along with un-cleared (or lowly cleared) maintainers led to a major Information Assurance nightmare. Methods currently do not exist to provide high assurance separation of the different security levels from System High systems.

Future avionics systems will consist of a large number of processors interconnected by LANs, fiber channels, and local buses. Avionics application software – navigation, flight controls, communication, displays, targeting, and weapons control – will operate in a distributed manner, with processes spread across thousands of processors. Humans will play a variety of roles in this environment including pilot, navigator, ground controller, ground support, and mission planner. There is also a trend toward autonomous vehicles, where there is no authority to supervise security decisions. The growing need to use multilevel systems in coalition environments makes this a "show-stopper" issue!

2.2 Joint Vision 2020 – System High Won't Work

Economics of general purpose computing has forced a tradition of developing software to share the processor resources. Operating systems, memory management, stack management, context switching, and interrupt vectoring are some examples of such sharing mechanisms. When avionics applications process different security levels of information, these sharing mechanisms must be trusted not to leak classified information between the processes. Trusted software is costly to develop, complex to design, poor performing, and it is difficult to certify its trustworthiness. As a result, most avionics systems avoid trusted development by operating at “System High,” the highest classification of any data entering the system. In the future world of the integrated battlefield, System High is not an acceptable solution. Weapon systems, sensors, and people will create multiple secure data streams at different security sensitivities, which must be managed in a MLS manner to permit battlefield flexibility of application of those assets, and not over-classify information to System High. We simply cannot clear all battlefield personnel to System High. High assurance MLS is a necessary requirement because of the hostile battle space environment consisting of data as high as Top Secret with multiple compartments serving friendly forces that will include uncleared, foreign coalition partners, Red Cross, and humanitarian personnel; the worst case threat environment by current standards [CSC003, CSC004].

2.3 MLS Problem

Simply stated, there are few Commercial Off The Shelf (COTS) solutions to satisfy the high assurance MLS requirements. The traditional alternative is to scratch build a high assurance trusted MLS system. That alternative is not attractive because 1) the avionics requirements are quite broad to meet all needs of the JV2020 battle space, 2) Certification and Accreditation (C&A) in DOD is in some disarray, with many competing approaches [DODD 8500.1& DODI 8500.2, NISPOM, DCID 6/3, DITSCAP, NIAP], 3) obtaining C&A is a lengthy process that may not complete by time of need, 4) systems may not satisfy real-time avionics needs, and 5) traditional MLS approaches are too expensive. A new approach is needed.

In July 1990, the National Security Telecommunications and Information Systems Security Committee (NSTISSC) was established for the purpose of developing and promulgating national policies applicable to the security of national security telecommunications and information systems. In January 2000, NSTISSC issued Policy No. 11, which addresses the national policy governing the acquisition of information assurance and information assurance-enabled information technology products. Policy No.11 states that information assurance shall be considered as a requirement for all systems used to enter, process, store, display, or transmit national security information. DOD has issued DOD Directive 8500.1, Information Assurance, and DOD Instruction 8500.2, Information Assurance Implementation, to implement Policy No. 11 [NSTISSC, DODD8500, DODI8500].

NSA and the Air Force have touted a trusted Protection Kernel (PK) as a candidate approach. They are supporting the development of a Common Criteria Protection Profile; a first step toward C&A [PKPP]. PK divides a processor into isolated domains with controlled inter-domain communication. A different security-level process can run in each partition. There is prior research encouraging this approach. COTS PKs available have weak security trust, and have never been applied to secure avionics application.

2.4 Moore's Law Predicts a Wealth of Processors

A modern aircraft today has over 1000 computers on board, and many more on the ground in support of the vehicle's mission. These are packaged into discrete systems with shared power systems, interconnect busses, and external communications. The computers perform flight control management, navigation, stores management, sensor processing, targeting, weapons control, communications processing, and display processing. Collectively, the high-level language software for these functions is multiple millions of source lines of code (SLOC), and rising with new developments.

For the past 30 years, computer hardware logic per chip has been growing exponentially, doubling every 18 months. First formulated as Moore's Law, the forecast in 2002 is for the exponential growth to continue with 12 logic doublings by 2020 (2^{12} = 4,000 X), our target timeframe [Moore]. The security challenge will be to build a high assurance MLS system from the expected array of 400,000 processors. We will be in an era of logic – processor/memory – richness. This paper proposes one way to securely organize and employ this computational richness.

2.5 MLS-PCA Characteristics

Key to avionics security is creating dynamic trusted connections between processes, not between processors as was achieved in the Defense Data Net (DDN) [BLACKER] and is typical with today's network Virtual Private Network (VPN) architecture. Cryptography today is placed at the junction of processors – host, router, server, firewall, and gateway – or within the processor with unknown quality encryption software, or as software mediated cryptographic chips. Alas the rub, all these approaches place the security base on untrusted software intermediaries.

Future computing will have processor- and memory-rich avionics designed as distributed processes in a plexus of processors interconnected by networks. Our approach is to move encryption to the process level to create trusted application connections with unique trusted cryptographic elements. The operative components of the architecture are an Encryption Process Element (EPE) interposed between an Avionics Application Process (AAP) and the communication channel. A Network Security Element (NSE) will control the Inter Process Communication (IPC) via distribution of encryption and authentication keys to the EPEs.

MLS-PCA implemented in a conventional network, would have an AAP hosted on its own processor. There is no need to share the processor and its resources with another AAP. There is no need for a complex resource manager or Operating System. A simple network protocol stack and loader is sufficient. Domains and domain management, e.g., context saving, context switching, are unnecessary. Memory management and sharing are eliminated as well as process scheduling. The absence of these features permits simpler hardware and CPU architecture, and the dedication of each processor to a single security level, that of its loaded AAP process. The elimination of processor and memory sharing is generally not typical for PCA chips. Chapter 5 of this report defines for the MLS-PCA model a Region of the 64 CPU PCA chip. The chip is divided into eight Regions. Each Region shares eight processors and/or memory cells; two for the EPE and six for the AAP. Within a Region processor and memory sharing is required. Each Region operates at the single security level of its loaded AAP such that there can be up to eight different security Regions – MLS -- sharing one PCA chip. However, between the eight AAP Regions of a 64 CPU PCA chip, there is no processor or memory sharing.

A small example of a MLS-PCA secure network is shown in Figure 2.5. Note the pairing of each AAP-EPE, including the NSE-EPE pair.

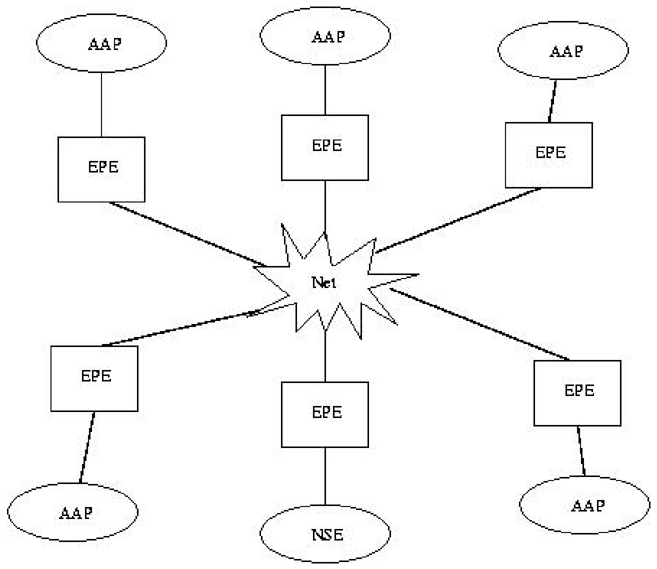


Figure 2.5: MLS-PCA Architecture Overview

3 MLS-PCA FUNCTIONAL MODEL

Avionics components are usually well defined by the mission and include air vehicle controls, navigation, e.g., Global Positioning System (GPS), inertial, targeting, sensor (e.g., Infrared, radar), weapons control, payload stores, communications, safety, and other systems. Ground support functions include maintenance and logistics, mission planning, mission analysis, and training among others. These support functions affect the avionics configuration. Mission planning determines flight plan, weapons, radio frequencies, crypto keys, weather, targets, etc. Plans so formulated are embodied in software programs and databases that are dynamically loaded into the air vehicle just before takeoff by some Portable Memory Device (PMD) carried by the pilot or crew.

3.1 Avionics Application Process, AAP

The avionics development includes infrastructure components – processors, busses, communications devices, etc. – under control of the appropriate application software processes. We define these as Avionics Application Processes, AAPs. Traditionally, AAPs are integrated into one large system operating at the system high classification of the vehicle, e.g., Top Secret Special Access Required (TS-SAR). MLS-PCA will require different thinking on the part of avionics developers. Functions will be classified individually at the single level of the highest data processed, often less than system high. Thus AAPs are the untrusted "subjects" of Bell-LaPadula [Bell], and will be at a variety of security levels, mostly Unclassified or Secret. Mission planning will select the required software for the mission, and construct a table – the access matrix – of the AAPs, which will specify their security levels, the data and devices, i.e., the "objects," they can access, and the type of access permitted, i.e., their read, write, append, and execute permissions. Furthermore, mission planning will define the avionics system configuration of network addresses, process ids, authenticators, and initial cryptographic keys. This classified data is protected from theft, unauthorized modification, and disclosure by encrypting the PMD for its journey from the classified mission-planning center to the classified air vehicle and back again after the mission with mission results.

An AAP is considered a homogenous process at a single security level. In reality, it may be many processes, but packaged for MLS-PCA as a single process. For real-time systems, an AAP traditionally is scheduled to run at a precise time interval by an event trigger, or by a call from another AAP. For MLS-PCA, the AAP will own its processor exclusively and need not be scheduled. It will always run (or be quiescent to save power), but only produce results when events dictate. When necessary, an AAP will interact with another authorized (by the access matrix) AAP. MLS-PCA will establish a cryptographically "trusted connection" between the two AAPs. Multiple AAPs can share a trusted connection as part of a "coalition." AAP trusted connections could last the entire mission, and often will in the well-defined world of avionics. Finally, the trusted connection can extend beyond the boundary of the avionics "box," or the air vehicle when properly configured. The trusted connection is only limited by the communications and imagination of the system developer.

3.2 Encryption Processing Element, EPE

Each AAP will be protected by an "attached" front-end guard element, the EPE. The EPE guards the attached process by performing message encryption/decryption of all IPC traffic. There is no bypass of the EPE. This is a security constraint on the architecture, the guarantee that a cryptographic computing element front ends each computational element. An EPE may be a software element or encryption hardware. There can be thousands of EPEs at any given time. An EPE does additional tasks related to protecting keys as a way of enforcing security policy. For example, all keys are distributed "wrapped," i.e., encrypted. The EPE must unwrap keys to use them. The wrapper key must be distributed in an "out of band" procedure, possibly carried in a physical "ignition" key generated by mission planning, and inserted into an avionics port by the pilot, or built into each EPE processor's nonvolatile memory by mission control. The choice is mission dictated and hardware configured.

In summary, each AAP has one EPE. The EPE is the only access between the AAP and the communications network and functions as a gateway to ensure that messages can be sent only to authorized recipients and that all messages are encrypted.

3.3 Network Security Element, NSE

The NSE distributes encryption keys to the EPEs, enforcing access control of communication paths, i.e., permissions between AAP pairs. The NSE is the security policy element for all internal and external communication, permitting the avionics interoperability with external battlefield assets. Within the control of the NSE is an access matrix of authorized permissions for each AAP. The permissions are stored as a database, with a unique key corresponding to each dimension of the security policy. For example, there can be a key for each security level and each compartment of the Mandatory Access Control (MAC) security lattice. There can be a common key for each user (uid) or process (pid) in a coalition, or a key for each AAP pair allowed to connect as part of Discretionary Access Control (DAC). There can be keys for each mission function, and there can be one-time session keys for each newly created trusted connection. NSE creates a trusted connection, by sending a session key to the attached EPEs. That session key is the XORed result of all the policy keys – the MAC, DAC, and other keys – for the connection based on the maximum authorized permission of the paired application processes. The NSE access matrix is authorized and established by mission planning and transported to the avionics system on a PMD at mission initiation. Dynamic updates are permitted by authorized roles in the mission, e.g., pilot, and/or ground control.

At mission initialization, system required trusted connections are established between security infrastructure elements – NSE, EPE (cf. Section 5.6.2). They exist to allow the NSE to distribute keys securely to EPEs. Information regarding AAPs is required for setting the NSE access database at mission initialization. A human role is defined by associating a user (uid) with a process (pid) in the access control matrix. For each pid and uid there is a set of credentials that defines the security permissions, the coalitions, and the roles played by all entities. There is a need for Identification and Authorization (I&A) whenever connections are established. The NSE will perform the I&A task inasmuch as it already has the I&A data from mission planning. The NSE can be implemented as a set of distributed processes executing on multiple processors within the avionics architecture for redundancy and performance, similar to any of the avionics applications

3.4 Security Policy Enforced by Encryption

The enforcement mechanism of the MLS-PCA model is the allocation of an encryption key for the trusted connection between two AAPs – the session key, $K_{session}$. The NSE computes the session key for each open request by an AAP to access another, based on the applicable security policy. Typically, there are multiple applicable policies – MAC, DAC, and Mission.

MLS-PCA treats AAPs as untrusted subjects, and treats trusted connections (TCs) as the security objects. TCs are simplex (unidirectional), i.e., AAP_i can write messages to AAP_j

(who reads messages from the connection). If AAP_j wishes to respond to AAP_i, AAP_j must open a separate simplex connection to AAP_i. Dialogs between AAPs can be “duplex” by creating two simplex connections. Simplex connections allow blind write-up, or Append, e.g., AAP_i may write to AAP_j, when the security level $SL_j >= SL_i$ (dominates).

Mandatory Access Control, MAC, is the classic DOD policy of a subject’s clearance dominating an object’s classification. This is best realized in the Bell-LaPadula [Bell] policy. MLS-PCA uses BLP and labels all subjects and objects. There is a MAC key, for each classification level, K_{sl} and each security compartment, K_{comp} .

Discretionary Access Control, DAC, further limits subject-object access. DAC is like a “wiring diagram” of mission functions (AAPs). DAC is conceptualized as a matrix of subjects vs. objects, with a matrix cell’s content containing the DAC encryption key, K_d. The DAC matrix is sparsely populated because the AAPs tend to cluster by function. For avionics purposes, a coalition is a collection of subjects who meet the requisite MAC requirements and are members of a community of interest of the MLS-PCA model. These subjects create a multi-party trusted connection by joining a coalition and leaving the coalition as necessary. MLS-PCA effects a coalition by treating coalitions as objects in the DAC matrix and creating a common key K_{coal} used by all coalition subjects. For each subject in a coalition, its coalition key, K_{coal}, is contained in the DAC matrix coalition cell. Thus, the DAC policy key K_{dac} is defined as: $K_{dac} = (K_d \text{ or } K_{coal})$, i.e., either the DAC key or the coalition key for a given object.

The MLS-PCA model is applicable to a wide family of avionics applications in a dynamic battle space environment. Missions can cover surveillance, targeting, shooter, and communications. MLS-PCA takes the view that an avionics mission is composed of a set of AAPs that constitute the mission functionality. The mission can then be represented by the DAC policy above. For multi-mission scenarios we need another (3rd) dimension to the DAC matrix that shows the DAC connectivity for each mission, i.e., another layer in the DAC matrix.

Overall then, the MLS-PCA security policy is reflected in the following:

$$K_{session} = K_{sl} \otimes K_{comp} \otimes K_{dac}{}^{1};$$
$$\text{where, } K_{dac} = (K_d \text{ or } K_{coal}) \otimes K_{mission}$$
$$\text{and } \otimes \text{ is XOR}$$

This scheme provides great flexibility in MLS-PCA to match security policy to the needs of the avionics application. Most missions are static with fixed AAP communication patterns as one might find in an autonomous Unpopulated Air Vehicle (UAV). In such a static environment, we might do away with the NSE and have access policy keys pre-placed during initialization at the EPEs by mission control.

1 Added security can be achieved by applying a non-invertable function to $K_{session}$ to foil a rogue process impersonating an EPE from obtaining $K_{session}$ and deducing the component keys.

3.5 Special Crypto Issues

The MLS-PCA model is silent on how the encryption function is mechanized – in software or hardware. It is only concerned that it be correct, always invoked, and always bound to its AAP. It is the "reference monitor" for the architecture [Anderson].

The model is also silent on the encryption algorithm to be employed. We only assume it will have management features compatible with DOD Type I and Type II encryption, and commercial algorithms such as Triple Data Encryption Standard (DES), and the Advanced Encryption Standard (AES). Choice will be made at the time of specific application. We do specify a Public Key Infrastructure (PKI) scheme for secure key distribution during system boot (cf. Section 3.6). Key management is intimately tied to security policy as discussed in Section 3.4.

Every secure system must have a means of revoking access upon discovering hostile, or runaway behavior of a subject. This means revoking a trusted connection immediately. Revocation is achieved by erasing the guilty AAP connection by "zeroizing" the session key for the connection, $K_{session}$.[2] Zeroizing is a command sent from the NSE to the EPE guarding the guilty AAP. Since the NSE and EPE are trusted processes the key erase action occurs near instantaneously breaking the AAP trusted connection. The AAP cannot thwart the zeroize action because it is not a party to the private infrastructure commands between the NSE and EPE. Also, unlike zeroize of traditional encryption boxes, the zeroize command can be acknowledged and states synchronized after action taken by the EPE, which has a separate trusted connection with the NSE. The model also uses zeroize of K_{coal} at a specific EPE to remove a subject (i.e., AAP) from a coalition.

3.6 Initialize and Bootstrap of MLS-PCA

The NSE and the EPE is the Trusted Computing Base (TCB) for the MLS-PCA scheme. There are two possible implementation configurations for the MLS-PCA model to protect the TCB: the first has the EPE in hardware; second, has the EPE as a loadable software process. Our view of the first consideration is the EPE process is a hardware subroutine of the CPU chip, somewhat like floating point hardware. We are looking at the proposed PCA hardware chips for MIT's Raw and Stanford's Smart Memories for how the model maps into the hardware. Generally speaking the hardware configuration is an easier initialization implementation because most of the initial parameters are "wired" into the hardware, e.g., network addresses, or process logic. The unique hardware initialization issues are resource allocation considerations when there exist lots of CPUs, memory, and buses on a chip, i.e., a Raw chip has 16 CPUs; Smart Memories has 64 CPUs. The software EPE initialization issues are classical security and integrity issues, the harder solution of the two configurations.

[2] "Zeroize" does not mean setting a key of all zeros. It means replacing a key with a random value not known by any other EPE, thereby making encrypted text using the zeroized key undecipherable.

3.6.1 Assumptions

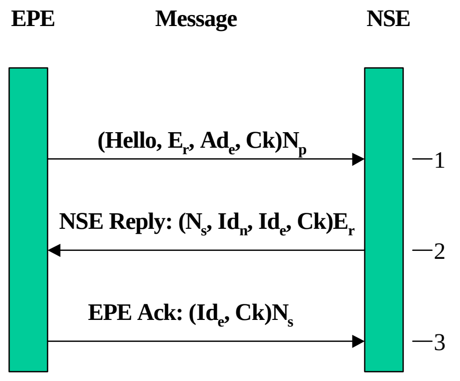
For any given classified avionics environment, the classified data and applications (AAPs) will be created and configured in a classified and trusted ground-based support system, a Mission Planning Center (MPC). The MPC is an MLS trusted facility that plans the mission, assembles the avionics mission software AAPs from trusted configuration files, and defines the mission configuration parameters (i.e., AAPs, NSE, EPEs, flight plan, radio frequencies, encryption keys, security levels of AAPs, weapons and fuel stores, and other items). The mission vehicle information systems will contain only unclassified data when "parked," be it an aircraft, UAV, or ship. The mission configuration parameters will be written to a PMD to be loaded into the vehicle just prior to the mission. The PMD will be encrypted to protect the pre- and post-mission information stored on the PMD.

There is a well known problem in trusted systems we call the "fixed point theorem." Encryption keys can be wrapped in other encryption keys, which can be wrapped in still other keys to a desired depth, for protection during transmission and storage outside of the crypto component. However, at some fixed point there needs to be a secret cleartext key pre-placed to permit decryption to begin to unwrap keys for the boot process to unfold in a staged and protected manner. In MLS-PCA the fixed point is a physical "ignition key" inserted into the system, and a pre-placed PKI private key in non-volatile memory of the NSE processor board. The ignition key is used to begin the unwrapping of encrypted keys using a physically protected token. To decrypt the PMD, the ignition key will be carried to the vehicle by the pilot (or mission commander for pilotless vehicles) and inserted in the cockpit prior to takeoff. The ignition key, like the PMD, is created by the MPC. We anticipate NSA will be responsible for the PMD encryption/decryption logic and wiring of the ignition key reader and PMD.

Typically, there will be one NSE and thousands of AAP-EPE pairs. The NSE may be redundant or distributed for reliability. The boot logic for the system will have the NSE loaded first, followed by prioritized EPE-AAP pairs loaded from the PMD. The mission will drive all the initialization parameters. MPC will determine the load priorities, locations of all devices and processes (i.e., their net addresses, Ad_n and Ad_e), their identifications (Id_n, Id_e), and the PKI private key and public key of the NSE (N_v and N_p, respectively). MPC will also build a table of permissions and classifications for all AAPs, the Bell-LaPadula access matrix, for the mission. Lastly, the NSE will know all these initial conditions by loading the access matrix from the PMD; the EPEs will know some of these data by parameter loading by MPC or NSE for each EPE-AAP pair code loaded – N_p, Ad_n, Ad_e, Id_n.

3.6.2 EPE-NSE Initialization Protocol

There can be a priority of operation of the mission functions reflected in the order of AAP initialization. The NSE will know that priority. For an AAP to run it must first be bound to an EPE. Since both AAP and EPE are software processes, there is no spatial association other than that they run on different processors. Also, there is nothing unique about an EPE; any EPE can be bound to a unique AAP. The NSE reads the PMD and creates an EPE, loading and/or assigning parameters to bind it to an AAP. The EPE is then executed while the NSE creates another EPE. The EPE's first action is to generate a random key (E_r). Since all EPEs are identical, E_r must be based on some changing system variable to avoid repeating E_r among different EPE invocations. Its next action is to create and send a Hello message to the NSE, giving the Hello message identification, its net address (Ad_e), random key (E_r), and an integrity checksum, all wrapped in the public key (N_p) of the NSE. This foils unauthorized reading of the Hello message by possible Trojans hidden in the architecture. The NSE saves these parameters and assigns the next priority AAP to this EPE by assigning an identity (Id_e) to the EPE; Id_e can be the identity of the bound AAP. It includes its own Id_n to confirm to the EPE its identity, gives a newly created NSE-EPE session key (N_s) based on the security level of the bound AAP, adds an integrity checksum, and wraps the whole message in the EPE's E_r. This provides critical information securely to the EPE to whom it is bound, including the session key for further NSE dialogs, the identity confirmation of the NSE, and an indication that a false NSE is not spoofing it. The last message by the EPE is an acknowledgement to synchronize state with the NSE. This complete initialization sequence and state space is shown graphically in Figure 3.6.


EPE
Message
NSE
(Hello, E_r, Ad_e, Ck)N_p
NSE Reply: (N_s, Id_n, Id_e, Ck)E_r
EPE Ack: (Id_e, Ck)N_s
1
2
3

State Space

	EPE	NSE
@Start:	N_p, E_r, Id_n, Ad_e, Ad_n	N_v, Id_n, Ad_n
@ End 1:	N_p, E_r, Id_n, Ad_e, Ad_n	N_v, Id_n, Ad_n, N_s, E_r, Ad_e
@ End 2:	N_p, E_r, Id_n, Ad_e, Ad_n, Id_e, N_s	N_v, Id_n, Ad_n N_s, E_r, Ad_e , Id_e
@ End 3:	N_p, E_r, Id_n, Ad_e, Ad_n , Id_e, N_s	N_v, Id_n, Ad_n N_s, E_r, Ad_e , Id_e

Where:

N_p = NSE Public Key **N_s = NSE-EPE Session Key**
E_r = EPE Random Key **Id_e = EPE Id, Id_n = NSE Id**
N_v = NSE Private Key **Ad_e = EPE Address**
Ad_n = NSE Address

Figure 3.6: EPE Initialization Protocol

4 MLS-PCA FORMAL MODEL

This section discusses the formal modeling effort [Hashii03a]. It begins with a description of the formal model specification language. It then describes the formal model, MLS-PCA constraints arising out of the formal specification, and experiences learned.

4.1 Formal Methods

Formal methods are the use of mathematical notion for specifying and proving theorems about programs. Frequently, the act of formal specification forces developers to think clearly about the design of a system.

Typically, specification alone is not sufficient, as one is not sure if it is right. Verification can be done via review by other formal methods specialists and domain experts. However, this approach can be arduous and time consuming. What is desired is something that provides more immediate and precise feedback. There are a number of other tools that are relevant. Type checkers provide syntactic checks as a first defense against incorrect specification. However, as more analysis is usually required, type checkers are typically part of a theorem proving or model checking system.

Theorem proving would be ideal, since it provides an actual proof that a specification describes behavior consistent with its correctness criteria. However, theorem proving tends to be a long and arduous process, usually requiring a user to create or direct a proof, even with automated tools. The goal for a theorem prover is, given a theorem, automatically generate a proof of its correctness. Since theorem proving is generally undecidable, automatic theorem provers cannot work in all cases. On the other end of the spectrum, proof checkers require the user to generate the proof. Most theorem proving tools lie somewhere in the middle, containing some automatic deduction while allowing the user to direct the proof steps.

Model checking, on the other hand, is completely automated. A model checker searches the system's state space for invalid states. Since one cannot possibly check all states, the state space is usually restricted, either by modeling the system as finite-automata or by restricting the range of the variables at analysis time. Similar to testing, model checking usually does not result in a full proof of correctness. However, it will give a good indication of a specification's correctness and is a useful middle ground between doing formal proofs and doing specification only.

4.2 Alloy

As mentioned earlier, formal specification is necessary for high assurance systems. We chose Alloy as an approach to performing formal analysis. This section will present an overview of Alloy. Alloy was developed by Dr. Daniel Jackson at MIT for the purpose of abstract software design. We begin by describing why we chose Alloy as our formal modeling language. We then describe the Alloy language and analyzer.

4.2.1 Choosing Alloy

Analyzing a formal specification by hand is beyond the abilities of even top mathematicians. As a result, we needed to find both a formal specification language and tools that would support it. We examined a number of formal specification language to use in our effort. We looked at Ina Jo, Z, EVES, PVS, Isabelle, ACL2, VDM, Alloy, and ASTRAL.

The criteria for evaluation of the methods include:

- Language: How understandable is the language. Is it intuitive? Is there support for mapping a specification onto an implementation, also referred to as refinement, i.e., can one prove that an abstract specification correctly represents the implemented system?) What are the ways of expressing state transitions? Are there ways to explicitly specify error conditions?
- Verification: Are there proof tools? How automated is the proof tool?
- Availability: Are the tools free for commercial use? If not, how costly are they? Is there support for the tools, either from a user's groups or the vendor? What is the level of documentation? How extensive is the documentation? Are there good examples? Is there an active mailing list or newsgroup? Has it been successfully used on other third party projects?

Our primary choice was Ina Jo [Locasso]; however, there is no longer any tool support for it. Others, such as ACL2 [Young] and Isabelle [Griff], are more concerned with proving theorems than creating easily readable state models. A language such as Z [Spivey] is ideal for specifying the sort of formal state machines required for high assurance systems. However, the difficulty is that many theorem proving tools, particularly those for Z (e.g., Z/EVES [Saaltink]), while good for academic use, were, at the time, extremely expensive for commercial users. For a more detailed survey on these languages, see [Hashii01].

Although we had originally desired theorem proving capabilities, a good deal of the advantage to using formal methods is in the writing of the specification. Writing a specification forces one to think through the design. Actually proving the specification, on the other hand, requires a much larger investment in time and resources. However, a specification alone is not sufficient, as the validity of the model remains uncertain. The alternative to theorem proving is to have an expert eyeball the specification for errors. We decided that automated model checking through state space exploration is a reasonable compromise.

As a result, we choose the Alloy constraint language from MIT [Jackson01a; Jackson01b; Jackson96]. The original version of Alloy was based on a combination of Z and UML. A newer version dropped the UML aspect in favor of more generality. Alloy has a freely available tool, the Alloy Analyzer (AA), that operates by logically enumerating the various possible states in an attempt to find an instance that fits, or violates, the constraints. In addition, since Alloy is based on first-order logic, the specification is more amenable to full theorem proving, should the technology become available.

4.2.2 Language features

The Alloy language is comprised of a fairly straightforward ASCII text notation. The language is based on set theory, similar to Z, with the standard set operators and quantifiers. A state is defined by sets and relationships among them. An operation will transform a state to a new state, i.e., the sets are modified. Alloy also allows the specification of invariants. AA will attempt to show that no operation produces an illegal state that violates the invariants. One can then use an inductive argument to claim that if an initial state is legal and all operations produce legal states, the system cannot be in an illegal state and the specification is correct.

4.2.3 Alloy Analyzer

AA is a tool that can be used to determine if a specification is over-constrained or under-constrained. One specifies a scope size, the number of instances of a particular type of variable. If no instances are found that satisfy the constraints, then the model is over constrained. One can also specify assertions that the analyzer attempts to disprove by finding counterexamples. In this manner, one can detect if the model is under-constrained. Note that this does not create a formal proof of the model, only that the model is valid within a particular scope. The possibility exists that the analyzer might be able to find a counterexample by increasing the scope size.

4.2.4 Translation to Alloy 2

While we were refining our model, the Alloy language underwent a revision. We decided to migrate our model to the new version of Alloy. The new version of the syntax introduced the notion of a signature into the language. A signature essentially defines a set, and is analogous to a variable type. State is generalized to a signature. Operations, then, take the pre- and post-states as parameters. Only the relationships that can change are part of the state signature, simplifying the frame condition. Typically when specifying state transitions, one must not only say what changes, but also what does not change. This is known as the frame condition. As one can write a state signature so that only the relationships that can change are a part of it, the frame condition can be greatly simplified. Signatures also increased the modularity of the specification, allowing us to analyze operations in isolation from other operations. Unfortunately, we were unable to modularize the state to the extent we hoped due to the interdependencies between the different MLS-PCA model elements. However, we did use the increased modularity to refine the model in those areas that could be isolated, resulting in a greater understanding of the architecture.

4.3 Model Description

This section provides a brief overview of the formal specification. Table 4.3 gives the size of our specification in terms of the number of elements that make up the MLS-PCA Alloy model: signatures, relations, operations, invariants, predicates, and facts. This section will discuss and give examples of each in turn.

Table 4.3: MLS-PCA Formal Specification Characteristics

Feature	Quantity
Signatures (i.e., Domains)	63
Relations (i.e., State Variables)	64
Operations (i.e., Transforms)	39
Invariants (i.e., Constraints)	18
Predicates (i.e., Conditionals)	38
Facts (i.e., Definitions)	28

4.3.1 Signatures

The model has five basic signatures or types: processors, processes, data, security labels, and state. The other 58 signatures consist of subtypes of these five. There are two primary types of processes: EPEs and AAPs. The NSE is a subtype of AAP. We also define another subtype of AAP, the SSO (System Security Officer), that has certain administrative privileges, such as changing DAC permissions and rekeying connections. There are three main types of data: messages, cryptographic keys, and audit logs. Messages are further divided into the individual message types used to implement the model's various protocols. Likewise, cryptographic keys are further divided based on their functionality: authentication keys, encryption keys, and policy keys. This last separation was to ensure that cryptographic keys are used for the purposes for which they are intended, and, more importantly, they are not reused for another purpose [NIST]. If a key is reused, say in an encryption and an authentication algorithm, then a weakness in one could increase the ability of cryptanalysis on the other.

4.3.2 Relations

The 64 relations between signatures consist of both dynamic relations, defined in the state signature, as well as static relations defined outside of the state. Examples of the dynamic relations include the binding of AAPs to EPEs, which processes are running on which processors, the contents of local memory and communication buffers, and current coalition membership. Examples of the static relations include message fields, both header and payload, that should not change throughout the life of the message. Another example is the definition of encryption, which is a relationship between a key, a message, and an encrypted message. The security labels of both processes and data are also static relations. We had considered making them dynamic relations, but that would have violated the Bell-LaPadula (BLP) Tranquility Principal [Bell; Sandhu]. BLP is our formal security policy for the MLS-PCA model.

4.3.3 Operations

The operations describe all possible state changes. Many of them reflect the model's protocols. Each protocol contains a dialogue between the NSE and an EPE. Each part of the dialogue has its own Alloy operation. For instance, the open protocol (shown in Figure 4.3.3) begins with an AAP, *A*, sending a message, *M*, to another AAP, *B*. The second operation has the EPE taking that message, realizing that it has no key for that connection, and sending an open request to the NSE, encrypted in the session key *SA1* that is used for communicating with the NSE. The third operation has the NSE's EPE decrypting messages and passing cleartext to the NSE. In the forth operation, the NSE determines if access is allowed, and either sends keys, *SAB* to both ends of the requested connection, or it replies with a denied access message and records an audit log. The fifth operation has the NSE's EPE encrypting and sending the *SAB* response to *A*'s EPE and *B*'s EPE. Note that because session keys are unidirectional, another NSE-EPE key for the other direction is used by the NSE's EPE, *SA2*. There are two options for the sixth operation. Either the AAP's EPE decrypts the response and has the keys or it is denied access. Assuming access is allowed, the AAP's EPE sends an acknowledgment that the key was received. When both acknowledgments are received in operation seven, the NSE will send a message to *A*'s EPE to start using the key. In operation eight, *A*'s EPE will send *M* encrypting it in the new session key, *SAB*. *B*'s EPE will now be able to receive and decrypt *M*, and finally, in operation nine *B* will receive *M*.

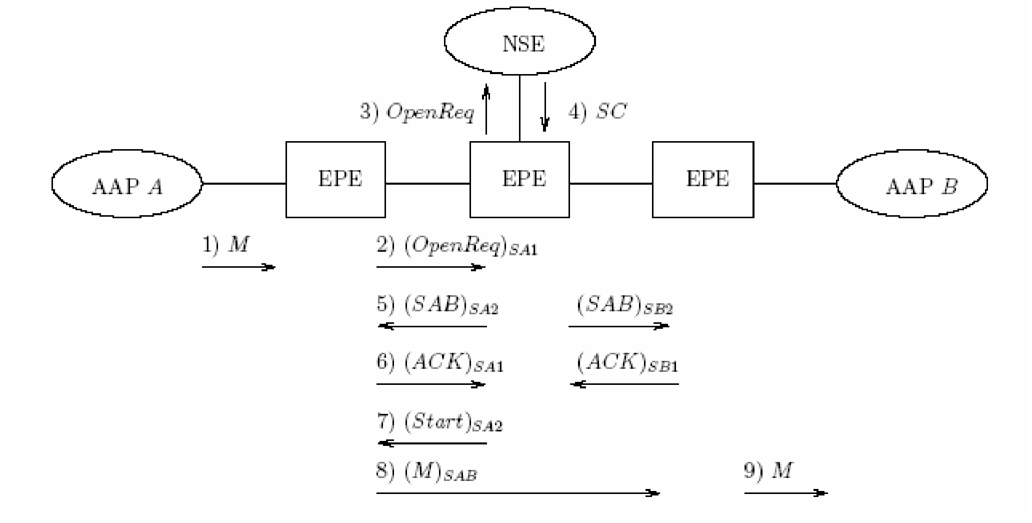


Figure 4.3.3: Open Protocol

There are two additional operations involved in this sequence that are not shown. One is to move messages around the network. The other decrypts and authenticates messages arriving at an EPE. There is a similar sequence of operations for each of the other protocols.

4.3.4 Invariants

Invariants are used to show the correctness of the model. The primary invariants pertain to ensuring the validity of the MLS BLP policy, i.e., an AAP's security label always dominates the label of data in its memory. Some additional invariants are required to ensure that the policy is valid after every operation. For example, the MLS BLP policy invariant also requires an additional invariant that says that an EPE only sends a message if the receiver is allowed to access it. As an EPE will only send a message if it has an appropriate key, this invariant, in turn, requires another that says that if an EPE can decrypt and authenticate a key distribution message from the NSE, then the EPE is allowed to use that key. As the NSE can guarantee the correctness of this statement, the original invariant can be shown to hold. In this manner, the functionality of the system can be shown to be correct.

4.3.5 Predicates and Facts

The final part of the specification includes predicates and facts. Predicates are helper functions that are used to ease the specification of operations and invariants. They provide a higher-level abstraction for a set of constraints. For example, there is an allowed access predicate that specifies both MAC and DAC constraints. There is also an encryption predicate that defines what encryption means in the model in terms of relationships between messages and keys.

Facts are constraints on the model, similar to invariants, but differ in that they are by definition true. For instance, there is a fact that says a message's sender defines the classification of the key used to encrypt that message. Our architecture does not check data security labels, so there is no reason for them to exist in any implementation. Instead, the data's security level is inferred from the level of the process that sends the data, with the exception of the NSE, which is a trusted MLS subject, operating at the security level of the connection receiver. The model associates a classification with data in order for invariants to be written that ensure data at a high classification level does not reach a process at a lower level, i.e., an unauthorized write-down.

4.3.6 Alloy Example

As an example of how the Alloy Constraint Analyzer provides feedback for modifications to the specification, this section presents one of the problems we found. In the specification, encryption keys are represented as data elements with an associated security level classification. The problem was that the classification of session keys was not initially constrained, i.e., the security level of the key must dominate the security level of the connection. The analyzer detected this problem during inspection of the `EPEReceiveKey` transformation, operation five of the open protocol illustrated in Figure 4.3.3. `EPEReceiveKey` is the transformation that details the process of an EPE receiving a key distribution (`KeyDist`) message. The case presented here is for the EPE that will be the receiver end of the new connection being created.

The operation `EPEReceiveKey` transitions from state `s` to state `s'` and describes what happens when an EPE (`p`) receives a key in key distribution message (***`KeyDist`***) from the NSE. The EPE essentially removes the message from its processing buffer `(Processbuffer)` where the decrypted and authenticated messages are stored for further processing, adds the key to local memory `(LocalMemory)`, creates an acknowledgement `(ack of type ACKReceiveKeyDistMsg)`, and sends an encrypted version of it `(dack)` to the NSE by placing it on its outbound network buffer `(Outbuffer)`. The transform is shown here with ellipses inserted for brevity:

```
fun EPEReceiveKey(s,s':State,p:EPE) {
  // precondition
  executing(s,p)
  some m:KeyDist | {
    m in s.Processbuffer[p]
    s.bound.p in m.receiver
    …

    // body
    some ack,dack:ACKReceiveKeyDistMsg | {
      ack.receiver = m.sender
      ack.sender = s.bound.p
      ack.KeySender = m.newCurrentKey.source
      ack.KeyReceiver = m.newCurrentKey.dest
      encryptAndAuthenticate(s,p,ack,dack)

      s'.Outbuffer[p] = s.Outbuffer[p] + dack
      all t: Process - p | s'.Outbuffer[t] = s.Outbuffer[t]
      …
      (s.bound.p in m.newCurrentKey.dest || s.bound.p in
            s.members[m.newCurrentKey.dest]) => {
        …
        assignedAddSub(s,s',p,m.newCurrentKey,m)
        …
        }
      }

    s'.Processbuffer[p] = s.Processbuffer[p] - m
    all t: Process - p | s'.Processbuffer[t] = s.Processbuffer[t]
    all t: Process - p | s'.previousKey[t] = s.previousKey[t]
    }

  // frame
  …
  }
```

The first step in analyzing the transformation would be to check that it is not over-constrained. In other words, given some valid starting state, e.g., Figure 4.3.6-1, the constraints specified by the transformation from that state to some second state, e.g., Figure 4.3.6-2, are satisfied. Alloy does this through the use of the *run* command. The *run* command allows us to simply search for a valid instance. In this case, the search was run over the logical *AND* of valid starting states and the transformation. For this transformation, instances were found immediately. Validity means a state and transform satisfy the model constraints and facts.

The second step is to check whether every possible instance results in a valid second state; this is checking that the specification is not under-constrained. Alloy does this by using the *check* command. Alloy *checks* the implication that a valid starting state and a valid transformation results in a valid second state by attempting to find a counterexample. In this example, Alloy produced a counterexample (shown in Figure 4.3.6-2), which shows a second state – Figure 4.3.6-2 – that violates the invariant `invMemAccess`. This invariant states that a process's security level dominates the level of all the data in its memory:

```
fun invMemAccess (s:State) {
      all p:Process,d:Data | inMemory(s,p,d) => canAccess(p,d)
      }
```

The problem here is that the initial state is not one intended to be valid. The classification of the key should be reflective of the connection that it serves. The fix was to add a fact to the specification that sets the classification to the security level of the sender, except when the sender is an NSE, in which case it should be set to the security level of the receiver:

```
all k:sessionKey | {
      k.source     !in     NSE     =>     k.classification     =
k.source.currentLevel
      k.source in NSE => k.classification = k.dest.currentLevel
      }
```

The valid start and second states with the fix applied are shown in the Alloy state diagrams of Figures 4.3.6-3 and 4.3.6-4.

Figures 4.3.6-1 through 4.3.6-4 show portions of the visual output provided by the Alloy Constraint Analyzer, greatly edited for readability. Each figure represents one state of the system. Each ellipse represents one entity in the system and the arrows represent relationships between the entities. For instance, in Figure 4.3.6-2, the EPE “Principal_3” is executing on the physical processor “Processor_0”, which has the key “Data_3” in its memory. Likewise, “securityLabels_0” dominates “securityLabels_1” much like “top secret” dominates “secret” in conventional classifications. In the ellipses the parentheses terms show what subset that entity belongs to (i.e. EPEs are a subset of “Principal”). For keys, “dest” and “source” relate the key to a connection from the AAP “source” to the AAP “dest”.

These figures show how the addition of a constraint, in this case a fact, can correct an under-constrained state. Before the addition, the state shown in Figure 4.3.6-1 was valid. After the addition, Figure 4.3.6-1 was not valid because it violated the new fact we added *“all k: sessionKey | {... .”* Figure 4.3.6-3 shows a valid state. As a result, when the transformation leads to the resulting state – Figure 4.3.6-4 – it is now also a valid state. By analyzing the code and the counterexample provided by the constraint analyzer, one is able to both find holes in the model and their solutions. This is a very helpful feature of Alloy; it not only provides a pass/fail assessment of the model, it also provides insight to the problem on failure.

The entire finished specification can be seen in [Hashii03b].

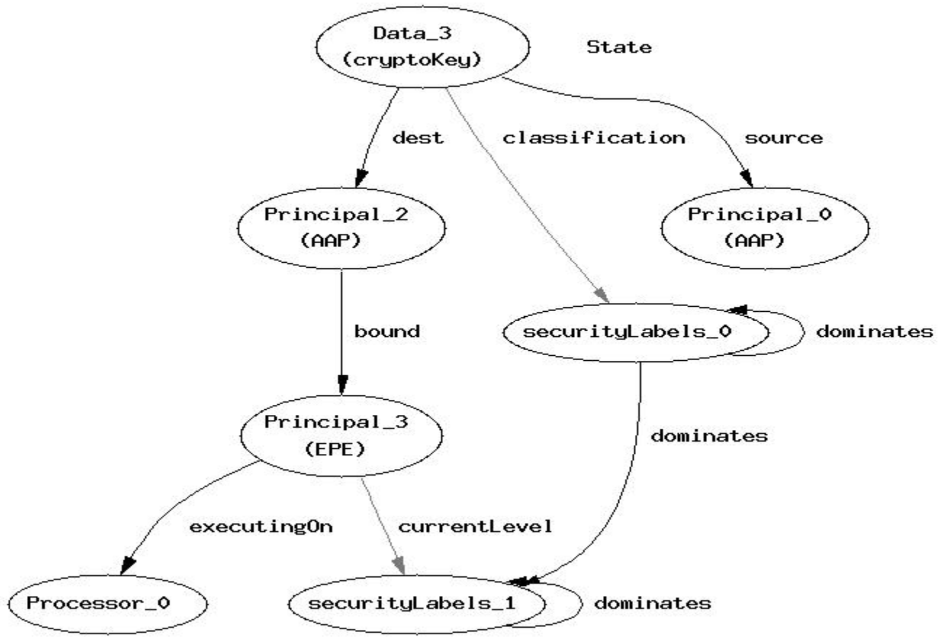


Figure 4.3.6-1: Initial State of Counterexample.

Notice that the classification of Data_3"(securityLabels_0) is higher than that of "Principal_3" (securityLabels_1) because securityLabels_0 dominates seculityLabels_1.

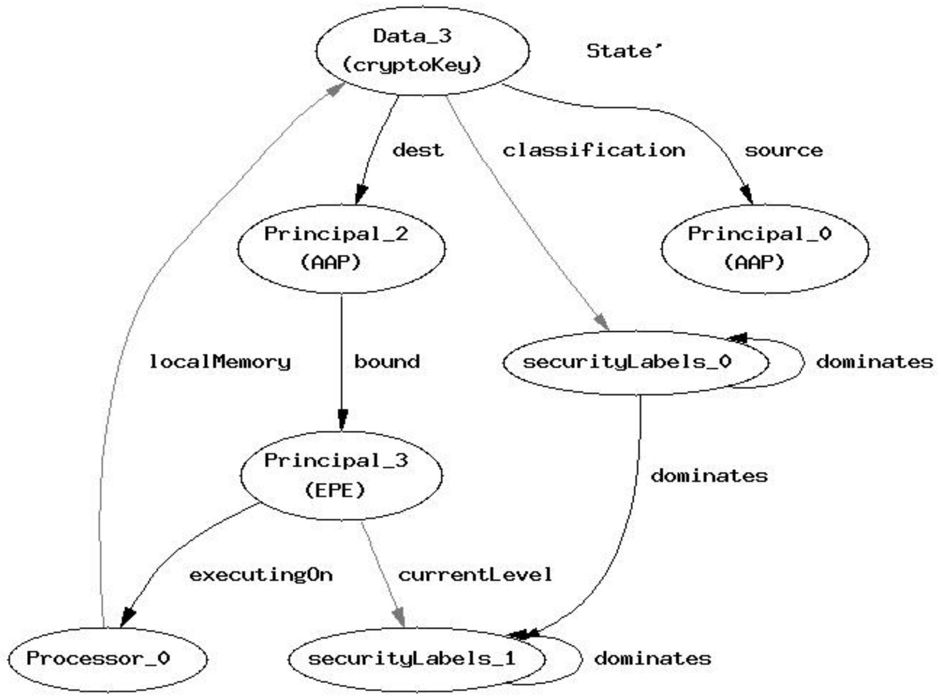


Figure 4.3.6-2: Resulting State of Counterexample.

Notice that "Principal_3" at securityLabels_1 is executingOn Processor_0 and has key "Data_3", at securityLabels_0, a higher classified level, in its localMemory. That violates the constraint *invMemAccess*.

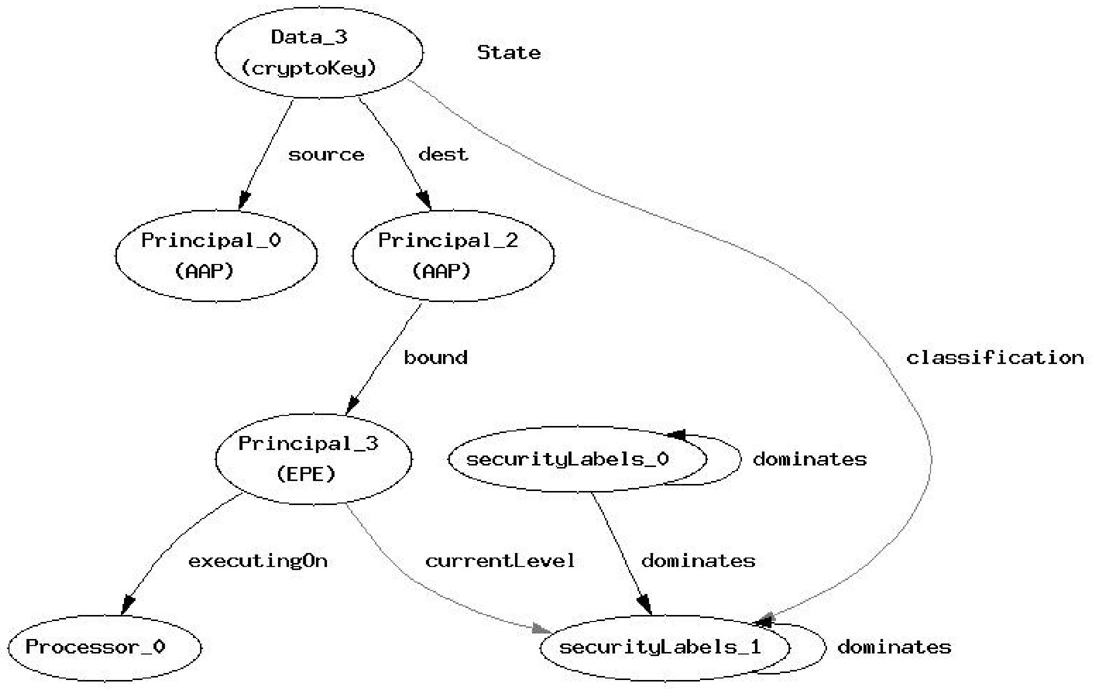


Figure 4.3.6-3: Initial State After Correction.

Notice that “Data_3” is now at the same classification level as “Principal_3,” securityLabels_1.

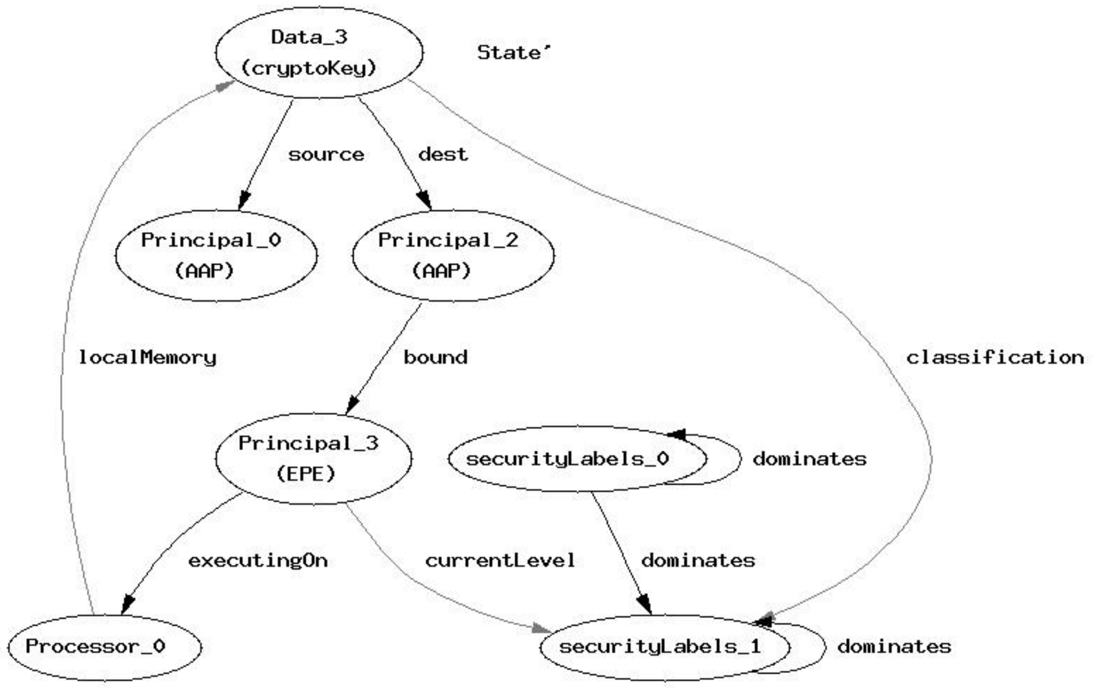


Figure 4.3.6-4: Resulting State After Correction.

Since “Data_3” is at the same classification as “Principal_3,” securityLabels_1, it can be added to memory without a violation.

4.4 Formal Constraints

One of the goals of this project is to determine the security constraints on PCA applications. Constraints show up in the formal specification in four places: the state description, facts, the operations, and the invariants. Table 4.4 lists these constraints.

The first area of constraint specification is in the definition of the state variables. For example, the constraint that there is only one process running on a processor is given in the state description of the relationship between these variables. Another example of this kind of constraint is that there is only one EPE bound to an AAP.

The next area of constraints is facts. These differ from invariants in that they are by definition true. For instance, the solution to the problem discussed in Section 4.3.6 is to have a fact that enforces the constraint that "the path's sender defines the classification of its key." Our architecture does not check data security labels, so there is no reason for them to exist in any implementation. Instead, the data's security level is inferred from the level of the process that sends the data, with the exception of the NSE, which is a trusted MLS subject operating at the security level of the connection receiver. The model associates a classification with data in order for invariants to be written that ensures data at a high classification level does not reach a process at a lower level, i.e., an unauthorized write-down.

Another example of a fact constraint is that messages must be received in order. Again, this is not something that our architecture checks, but is something that our architecture assumes is true of the underlying network.

The above two types are the ones most likely to place constraints on the PCA hardware or morphware. The next two types of constraints are ones primarily placed on our security architecture. The first of this type is the operations or transforms. A transform describes the condition that must exist before and after an operation. The notion of an AAP and its bound EPE sharing the same fate, "fate sharing," is made in the transforms dealing with starting and stopping a process. Similarly, a transform describes the constraint that an AAP can only communicate through an EPE.

The last type of constraint is the invariants. These are statements that the analyzer must show are true, and that the functionality of the architecture does not invalidate. Examples of this type of constraint include enforcement of the MLS policy that AAPs can only send data to other AAPs whose security level dominates their own, and that AAPs do not receive data that has a higher classification than the AAP's security level. Another example is that messages from the NSE are correctly formatted and can be authenticated as coming from the NSE.

Table 4.4: Formal Constraints

4.4.1	**State variables**
•	There is one process (AAP or EPE) per processor (Region).
•	There is one AAP and one EPE for each Region; they cannot share resources.
•	There is one EPE per AAP.
•	Each processor has local memory (for storing keys, messages, audit logs, etc.).
•	All buffers are in local memory.
4.4.2	**Facts**
•	The path’s sender defines the classification of its key.
•	Messages must be received in order.
•	The DAC key for accessing a coalition is required to be the same for all members of that coalition, so that a single session key is generated when the DAC key is used.
•	All EPE-AAP pairs can talk to the NSE
•	AAPs share an address with their bound EPE. Thus, all messages are sent through the EPE.
•	Messages do not encrypt to themselves, i.e., there is no “null” key.
•	Encryption and decryption do not change header information.
•	Encrypted messages are unclassified.
4.4.3	**Transforms**
•	EPE/AAP are fate sharing.
•	An AAP can only communicate through the EPE.
•	The NSE knows where the EPEs are.
•	EPEs don’t process control messages unless they are from the NSE.
•	Session keys are derived from mission keys, MAC keys, and DAC keys.
•	EPEs have at most two keys for a path, one current and one old, to enable rekey asynchronous with current message operations.
4.4.4	**Invariants**
•	Data is only accessible by a process if allowed by MAC.
•	If an EPE has a key, then the process is allowed to communicate in that direction.
•	EPEs only have session keys for processes to which they are bound.
•	Messages used by the NSE to communicate keys to an EPE are correctly formatted.
•	The process's security level is dominated by the environment’s level.
•	An EPE only sends messages that are allowed to be accessed by the receiver.
•	All EPEs bound to a process are executing.
•	All executing AAPs are bound to an EPE.
•	EPEs are bound to AAPs in a trusted manner, i.e., have trusted path.
•	Data in a processor’s memory is dominated by the processor’s level.
•	If an EPE has a key, then the EPE’s level dominates any data it receives using that key.
•	The key classification is the same as the information encrypted and sent using that key.
•	AAPs will only have message sent to them; EPEs will not send data to an AAP not meant for it.
•	All rekeys for a simplex path are only sent to members of the path.
•	The private key is only in the memory of the NSE and/or its bound EPE.
•	Unbound EPE’s only contain the public key of the NSE in its memory.

4.5 Influence on Design of Formal Methodology

The promise of formal methods is to detect errors early in the design cycle. The later one finds mistakes, the more costly they are to fix. This section will discuss some of the influences the formal modeling had on the design. The process of writing the formal specification resulted in early design decisions. AA found other problems. This section will first examine each area where formal modeling influenced the design.

4.5.1 Focused Design

The process of writing the formal specification focused the design of the functional model and raised many questions.

A big question was: How do you initialize the system? In other words, what is the initial condition? An early version of the formal specification asserted the initial condition and only dealt with the normal steady state operations. However, subsequent revisions forced us to determine how we arrived there. The NSE needs to be able to communicate with EPEs and it needs to be able to do so securely in order to distribute keys. Since these keys should not be sent in the clear, they must themselves be encrypted. This means that the NSE and EPEs must already share a key. How is this key distributed? The key could be built into the EPE code, but then how is this code distributed and loaded in a secure manner? There must be some bootstrapping mechanism to get a key loaded into an EPE.

There are a number of options we considered. The first was to have an ignition key physically inserted in the NSE and each EPE. However, while this might be acceptable for the NSE, as our model can consist of tens of thousands of processors, manually inserting a key into all of the EPEs is not feasible. Another option was to have the key built into the hardware. The Trusted Computing Platform Alliance (TCPA) has begun work involving built-in hardware keys [TCPA]. However, this approach suffers similar problems as there will be thousands of EPE processors. We wanted our approach to have greater flexibility. EPEs may exist in software and/or hardware, and we want to map the model to existing processors. We considered using Diffie-Hellman [Diffie 76] but that is susceptible to a man-in-the-middle attack. This form of attack is usually countered using certificates and public key signatures [Diffie 92]. The problem with the signature approach is that it requires the EPE to have a private key, which returns us to the problem of getting a private key into thousands of EPEs.

The solution we settled on came from the realization that, while we could not load the EPE with a private key, it could be loaded with a public one. Thus was born the solution discussed in Section 3.6. By forcing us to look at the problem early and often, the formal methodology helped us to arrive at a solution that works.

4.5.2 Design Error

Another advantage of doing a formal model is the ability to examine the model for design errors, even before doing a formal analysis.

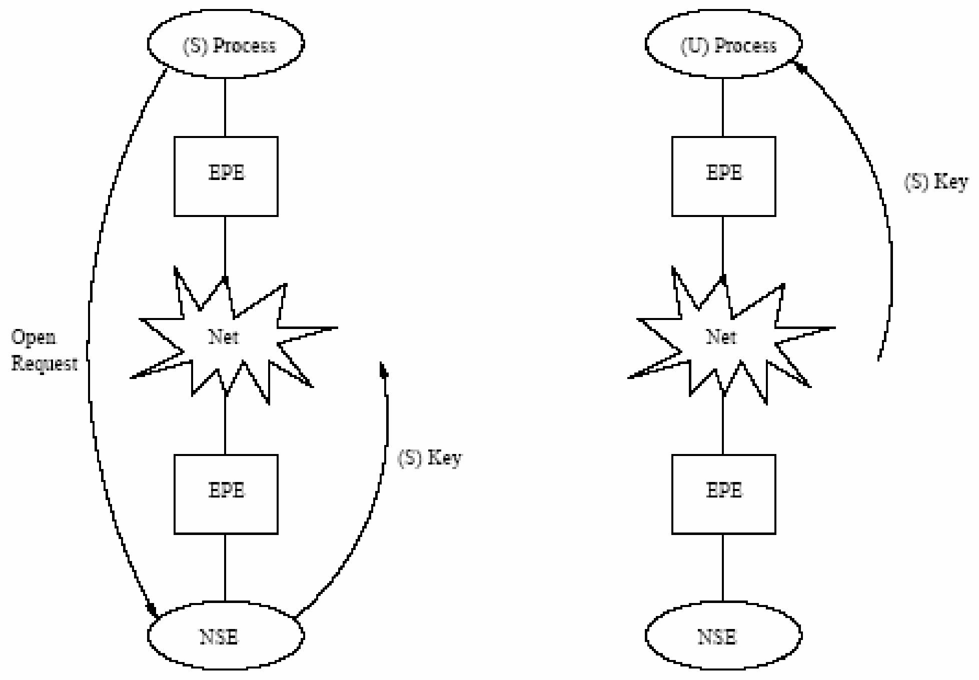
One such error was found when dealing with reboots. There are times when an avionics system needs to accommodate an inflight reboot [Tiripak 2002]. For example, fault recovery might consist of rebooting a corrupt processor. As explained in Section 4.5.3, when an AAP goes down, its associated EPE must also go down. As a result, all connections previously established will be lost. We could either notify the other end of the connection, or simply re-establish the connections. We chose the later approach. When an AAP/EPE combination comes back up, the EPE will do the initialization procedure described in Section 3.6. The NSE will then send a new set of keys for each previously opened connection involving that AAP.

Human examination of the formal specification found an error pertaining to the synchronization of the new keys. There are two types of messages used to transmit keys from the NSE to an EPE: the initial *key distribution* message and a *rekey* message. The key distribution message is the result of the open protocol of Section 4.3.3. Rekey is required when a crypto period expires or when other events demand a key change. The rekey protocol is similar to the open protocol illustrated in Figure 4.3.3, except that rekey doesn't require storing *M* until the protocol is finished as the EPE can send *M* using the old key. Rekey also uses a two key scheme. This means that the old key is stored in case a message is later encountered that is encrypted in the old key. Also like the open protocol described in Figure 4.3.3, the new key is not used until the EPE receives a message from the NSE letting it know that both sides have received the same key. In the case of reboot, the initial idea was to send a key distribution message to the rebooted EPE and a rekey message to the other side of the connection. However, the acknowledgments for a rekey and a key distribution message are different. The NSE, upon receiving both acknowledgements will not be able to tell if they are the result of a rekey due to reboot, at which point both ends of the connection will have the same key and the EPEs can start using it, or if the acknowledgements are due to separate open and rekey operations, at which point each end of the connection will have different keys. Thus, the message to start using the key will never be sent. In the rare case where a key distribution message is immediately followed by a rekey, the acknowledgment difference is necessary. The solution was to send a key distribution message to both sides and change key distribution to also keep track of the two key scheme.

4.5.3 Error Found Via Automated Analysis

By formally specifying the model in Alloy, we could check for errors using AA. The vast major of errors found by AA were errors in writing the spec: typos, an incomplete frame condition, and miscommunications in learning Alloy reflected in the difference between what was meant and what was said. However, occasionally an error in the formal spec will reveal an error in the functional design.

One of the earliest errors found by the Analyzer was the need for an AAP and its bound EPE to be *fate sharing*. In other words, they need to stop execution at the same time. Early versions of the specification without this feature had a problem. The spec allowed a process to die and be replaced by another AAP at a different security level, but bound to the same EPE. The analyzer found a case, illustrated in Figure 4.5.3, where a message destined for an old Secret (S) process could arrive at a new Unclassified (U) one. The problem is that messages should not be decrypted by an EPE bound to processes that are no longer running. Thus, an EPE and AAP should both go down at the same time -- sharing the same fate. This has the additional advantage of flushing the EPE's memory.



(a) A Secret bound EPE requests keys for a new connection and the NSE sends the (S) keys, but the EPE does not yet receive them

(b) The AAP reboots at Unclassified and then receives the (S) keys from the NSE

Figure 4.5.3: Fate Sharing

4.5.4 Design Simplifications

In our efforts to reduce the complexity and size of the specification, we discovered a number of areas where the design itself could be simplified.

One of these simplifications is the notion of a current security level and a maximum-security level. BLP distinguishes between a user's current security level and a user's maximum-security level [Bell]. The reason is that the user might at times wish to create and release information at a lower level than the user's maximum security level. Running a lower level process typically does this. Although it makes sense for a user to have both a current and maximum level, it makes no sense for a process to have them. Early versions of the specification contained both. However, since there is no explicit concept of a user in the MLS-PCA model the maximum level was never used in any meaningful way. As a result, it could be safely discarded.

Another simplification involved the revocation operation. The original idea was that revocation would take advantage of the rekey capability. When a connection was to be revoked, one side would be rekeyed with a fake key and the other would not, thus breaking the connection. In addition to needing to manage a fake key, we also need to make sure that we did not inadvertently perform a valid rekey and re-establish the revoked connection accidentally. While writing the specification, we realized that we have another operation with somewhat similar properties called *zeroize*. It essentially causes the EPE to zeroize all keys, and effectively shut down. A simpler implementation of revoke is to use the zeroize mechanism, but to only zeroize keys for a particular connection, instead of zeroizing all of them. Note that here zeroize does not mean to just set the key to zero, but also to no longer use the key, as we do not want to continue using a zero key. Thus, we were able to greatly reduce the complexity of the revoke operation.

4.6 Experiences Using Alloy

This section discusses our experiences in using Alloy to model our architecture. First, we will examine some of the limitations of our model. Next we will discuss our experiences using Alloy's graphics rendering features. Last, we will examine our experiences in creating an implementation based on the specification.

4.6.1 State Space Size

Our model is fairly large, as can be seen from Table 4.3. As a result, the greatest problem we encountered was the size of the state space. We performed our work on a 300Mhz, 128 MB Sun UltraSPARC-II running Solaris 2.6 and a 2.26 GHz, 512MB IBM Pentium 4 running Windows 2000. If the state space were too large, our development machines would run out of memory, resulting in the analyzer either crashing or hanging.

The modularity mechanisms introduced in the new version of the language helped tremendously. We could split each operation into modules. This significantly decreased compile time. However, modularization only resulted in modest decreases in state space. Each operation required the same state signature. Although there were occasionally some subtypes, for example certain message types that are only needed by a particular operation, the basic types are needed throughout. As a result, we were constantly walking the line between models that could and could not be analyzed.

The need to keep the model as small and simple as possible is an advantage in that it required a degree of model simplification and abstraction. For example, we assumed reliable traffic. The abstract formal model should not specify how a network is made reliable, only that it is. This allowed us to not require acknowledgments after every message, as one would find in TCP/IP; we only needed to synchronize state between the various elements. This also allowed us to forgo timers and, as a result, an explicit time variable. Likewise, encryption is a simple relation between a plaintext message, a ciphertext message, and a key. The details of the encryption algorithm, e.g., DES or AES, do not need to be specified.

However, there were times that the need to reduce the state space resulted in specifications that were not ideal. For example, consider the case of a revoke message being sent immediately after a key distribution message. There is a possibility that the EPE receives the messages out of order. In such a case the EPE might accept a key for a revoked path and thereby allow unauthorized communication. Thus, messages must be delivered in order. This is fairly easy to do using sequence numbers. Unfortunately, this would have increased the state space by adding a new signature type. Likewise, Alloy provides an ordered list package that could have been used for the communication buffers. This too would have increased the state space. We eventually decided to modify the definition of the message buffers so that only key distribution messages can be sent more than one at a time. This forces an ordering in that a message must be processed before the next is sent. This solution has the additional benefit of forcing a smaller state space. However, it adds an unnecessary constraint to the model. However, since an operation can only process one message at a time, we felt that this constraint would not prevent finding bad states.

4.6.2 Revocation Synchronization

The lack of time discussed in the previous section can, however, result in problems. For example, the NSE and EPE need to synchronize when DAC permissions are revoked or an AAP is removed from a coalition. One of the invariants the analyzer checks says ``a communications path between AAPs is in existence only if the MAC and DAC policies allow it." The problem is that DAC permissions can change and there is a delay in communicating that change to the EPEs. As a result, there will be a period of time where a path is in existence and it is not allowed. Most operating systems do not revoke access immediately because of the complexity involved in dealing with this synchronization issue. The Multics operating system, on the other hand, did implement immediate revocation [Karger]. However, the analyzer noticed that, unlike Multics, our revocation couldn't be immediate due to the distributed nature of our architecture. However, we make an attempt to revoke as soon as possible.

The end result is that the analyzer found a case where the operation for changing permission invalidated the DAC invariant. Normally, this sort of invariant check is used to determine if an operation is under-constrained. An alternative possibility, as was the case here, is that the invariants are not valid and it is they that are over-constrained. The solution was to weaken the DAC invariant and return to the policy that most operating systems have: only guarantee DAC is true for new paths.

4.6.3 State Diagrams

Alloy also has a feature for graphically representing the instances discovered by AA. We did not find this feature to be terribly useful. In fact the graphic typically produced by AA was unreadable, mostly due to the large size of our state space.

Although not useful for analyzing the model itself, the diagrams do have a slightly higher utility for illustrating a state instance for explanation to others. In order to accomplish this, the graphic needs to be pared down to contain only those variables and relations needed for the point being made. Fortunately, AA has a built-in facility for doing precisely that, as well as adding additional subtype labels.

However, the resulting graphic is still somewhat difficult to understand. Fortunately, AA uses a graphics tool from AT&T Labs called *dot* [Kouts], and modifying the resulting *dot* file is quite simple. There were two types of manual modifications that we made. The first was to remove unnecessary instances of a type. The second is due to differences between the current version of Alloy and the previous one. The previous version of Alloy produced two graphics, the before state and after state. As the new version of Alloy does not require state variables, AA now only produces one graphic with both the before and after states on it. We found it useful to separate back out the different states in order to better illustrate state transitions.

Despite these modifications, the resulting graphics were still not very enlightening for others unfamiliar with the language. However, we suspect them to be more enlightening for the uninitiated than the textual representations of the same instances. The results of these modifications can be seen in Figures 4.3.6-1 through 4.3.6-4.

4.6.4 Experiences During Implementation

After having corrected the problems discovered in Section 4.5, we began implementing a prototype using IRAD funding in order to further shake out the model. As we do not have thousands of processors on which to test, we simulate the environment using grid computing. The PCA paradigm assumes multiple CPUs per chip. This implementation process helped to further refine the design. These areas can be put into two categories.

The first contains cases that resulted from omissions in the operations. Although these omissions did not violate the security invariants, they are omissions that need to be addressed in the running system. For example, when a process reboots, the NSE will automatically re-establish the pair-wise connections, but not the coalition keys. The NSE either needs to distribute those keys or remove the rebooted process from the coalition. We decided that since the AAP is responsible for joining coalitions, it could be responsible for re-joining after a reboot. Thus, the preferred solution is for the NSE to remove the rebooted AAP from coalition membership until a new join request is made.

Another example is that the `NSESendRekey` operation only considers pair-wise connections, not coalitions. Note that in both case, there is no security invariant violated. The reason these problems were not detected is that we did not write invariants to cover these cases, as we had not thought about them at the time.

The second class consists of problems that are due to functional or operational errors. For instance, the model says that the EPE and its bound AAP should start and stop at the same time. In reality, the EPE will probably have to start first in order to facilitate the loading and execution of the AAP. Note that the *fate sharing* solution discussed above is only necessary for stopping executing, not starting. Another example is that the model treats an NSE-EPE rekey the same as any other rekey. When a rekey occurs, the new key is distributed to the EPE via the NSE-EPE session key. However, a rekey should not be encrypted in the key it is replacing. Thus, the NSE-EPE session key needs to be rekeyed using a different mechanism. The solution is to reuse the bootstrapping mechanism for setting up the initial session key.

Although there were a few areas where the implementation uncovered problems, the formal model provided a framework from which to analyze the problems and devise solutions. As many of the problems were found during the design, the implementation required very little re-coding.

5 MLS-PCA MODEL MAPPING TO PCA HARDWARE

This section of the MLS-PCA Final Report analyzes how (and how well) the elements of the MLS-PCA formal model map onto the PCA chip technology. As we will see, it is not a perfect fit, but it does appear to be feasible with some modification to the PCA development approach.

In much of the PCA development efforts, researchers have attempted to virtualize the various PCA architectural approaches into a common compilation target, e.g., see [Horowitz]. This has been a very useful effort. However, in the specific tasking being discussed here, we have done just the opposite. We have tried to determine the various hardware details that best allowed us to provide a certifiably secure computing platform in the specific area of MLS avionics processing.

We have utilized the MIT Raw [Agrawal, Taylor 02] and Stanford Smart Memories [Mai] PCA chips as the principal basis for our investigation since they are well documented in the open literature. Actually, we have developed a "meld" of these two approaches (with a few extensions to provide support for the cryptographic requirements of MLS-PCA and to extend the size of a processor group beyond a quad), which has produced a generic PCA chip for this analysis. A good survey of alternative PCA architectures can be found in Section 3 of [Campbell].

The various PCA chip developments were intended to exploit the benefits of parallelism by compiling a parallel-capable program onto the multiprocessor PCA chip(s) using whatever form of parallelism best fits the program's needs. We are attempting to exploit the multiprocessor capabilities of PCA in quite a different way. We plan to replace the several hundred pounds of liquid cooled super computer modules in a typical DOD avionics system with a handful of PCA chips. But, even more importantly, we provide a solution to the "MLS risk problem" that has plagued such systems for over a decade.

5.1 A Generic PCA Chip Architecture

Our generic PCA chip is a grid or matrix of computing elements, referred to as "tiles". Each tile may be a processor, an 8 MB block of Dynamic RAM (DRAM) memory, a combination of processing and memory, or a special purpose device (e.g., a crypto engine). Our "basic building block" is a 2 x 4 tile rectangular region, which can be an AAP/EPE or an NSE/EPE. The EPEs all connect to the global bus/network at the edge of the PCA chip. Only cipher text appears on this shared global bus/network. There is a per-region dedicated set of bus "wires" which connect the tiles within each region for the plain text traffic. Each region provides the morphable multiprocessor platform for one application process, e.g., electro-optical or radar processing. There would be eight such regions on one 64-tile PCA chip. Note that the target subsystem for the compilation is a region rather than the entire PCA chip. Region-by-region morphability is also required. The layout of a region is shown in Figure 5.1 below.

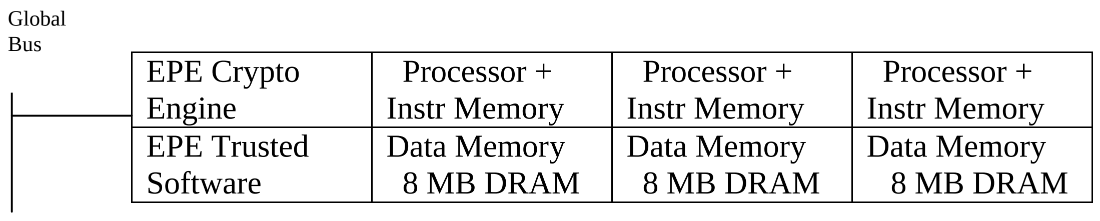


Figure 5.1: Fundamental MLS-PCA Eight-Tile Region Building Block

The EPE crypto and processor tiles are positioned at the two end tiles in the 8-tile region. They are at the edge of the chip, and the cipher text bus is connected to the global bus/network. The remaining tiles consist of processor and 8 MB DRAM memory tiles for the AAP (or NSE).

One of the fundamental reasons for having both processors and memory built into the same chip is the "wiring problem". This traces its roots back to the early days of super computers, when the interconnect wiring was a major limiting factor. Putting the wires on the chip (with VLSI) seemed to solve that problem. However, at today's processing speeds, we are back to worrying about the speed-of-light transmission times. Long wires are a limiting factor, but now "long" means more than one tile in length. Smart Memories stretches this to the size of one quad, and we have stretched that to two adjacent quads in length [Ho].

Our model of operation most closely reflects the Smart Memories bus/network configuration with some assumptions about how it could be "locked down" for accreditation purposes. In contrast, the Raw chip is quite different in this regard, and to quote their documentation [Taylor 03], "the static routers collectively reconfigure the entire communication pattern of the network on a cycle-by-cycle basis", usually a security "no-no".

However, since the avionics MLS-PCA will be a special purpose PCA chip due to the encryption tiles, we could etch away in hardware any possible inter-region paths between the plain text portions of the regions. With that boundary "nailed down", we could then let each region morph with impunity (including the above mentioned Raw routers). The crypto would likewise be built-in between the red and black buses with no morphing supported or required in that regard. Similarly, the AAP could not morph any aspect of the EPE operation.

The physically isolated regions present an interesting situation for the morphware compilers. In the typical avionics application, each separate region is the target machine for a different avionics program, e.g., electro-optical versus radar versus fusion processing. However, in a more general usage of this regionalized PCA chip, the total set of physically isolated regions could be the target machine. They could inter-communicate over the encrypted paths. If any-to-any (single level) communication is desired, the coalition key approach could be used, providing a common key for all regions in the set. This might be similar to today's Grid computing. Off-chip (EPE-fronted) global storage could also be included for the set (in the form of Network Attached Storage). So, while the "nailed down" region constraints constitute a new set of concerns with morphing, they are still consistent with the virtualizations of morphing with the Stable API (SAPI) programming model and the Stable Architecture Abstraction Layer (SAAL) hardware model [Richards].

There would be considerable benefit in the initialization and on going monitoring of the PCA operation if one provides a support processor to manage the PCA chip. The support processor would consider the PCA chip to be a co-processor. Another benefit in introducing the support processor is the need for some degree of control of the morphed configuration that may be essential for the accreditation process, which requires that the hardware/software configuration be well defined and unchanged (in any unexpected way) without reaccredidation. The support processor could at least monitor the configurations that are being morphed by the AAP controls by periodically reading back the morph configuration registers and comparing their content with a set of allowed morphs.

PCA chips generally have on-chip high speed SRAM for use as cache(s), and rely on DRAM chips connected to the global bus/network for their main memory. This does not work for the MLS-PCA situation due to the multiple levels of security that are involved. Each region would need dedicated "wires" and DRAM chips. Up to 1 GB per AAP may be required in the future, so some form of off-chip DRAM will be required, but must be accessible without having to use a shared bus/network. This might involve morphing a wide shared access bus/network into a number of dedicated more narrow bus/networks.

There is another memory aspect that is troubling, namely all of the on-chip PCA memory is volatile. Non-volatile storage is needed for bootstrap initialization and configuration storage as well as for a trusted distribution crypto key. Non-volatile memory is generally incompatible with processor and non-volatile RAM application and fabrication, requiring different voltage levels and/or additional masking layers. However, it is possible to build such chips, e.g., in "system on a chip" products [SoC NV], so we have not given up in this quest.

5.2 Related Technologies

There are several other technologies that relate to PCA that were found useful in this analysis. They are briefly described below.

PCA chips may have a considerable similarity to Field Programmable Gate Arrays (FPGAs), i.e., at least those with SRAM configuration capabilities. In both cases, the intra-chip wiring between the candidate set of resources is configurable. Our analysis has relied on the methods used in FPGAs (especially those of Xilinx) as one model of operation for the dynamic morphing process [Xilinx].

Some of the capabilities that we require do not fit 100% with the CMOS technologies that we would expect to be used with PCA chips, e.g., the desire for some amount of on-chip non-volatile memory (such as flash memory). In this example, the voltage level that needs to be applied to write flash memory is not consistent with conventional lower voltage level CMOS fabrication. Additional mask layers would also be required. We have relied on System-on-Chip (SoC) developments to judge if such MLS-PCA requirements are impossible, and we have found that mixed process technology is indeed possible, but may have other implications (such as additional costs or a sacrifice in logic density) [SoC NV].

While the EPE itself does not require MLS, it still does require security assurances such as "Trusted Distribution" of its operational software. The desired approach for trusted distribution is to have a copy of the source's public key built into the EPE hardware (e.g., in non-volatile memory). The load modules could then be authenticated using the integrity check that has been included with the software and signed by its source. There is also a need to have the public key of the NSE in non-volatile storage in each EPE for initialization purposes.

Another related technology is the Advanced INFOSEC Machine (AIM) programmable crypto chip that was originally developed by Motorola and is now a product of General Dynamics [AIM]. Our concern was if it would fit into the tile structure of a PCA chip. The publicly available information about the AIM chip is that it originally utilized a 0.35 micron fabrication process and contained 8.5 million transistors. The die size was not specified. We estimated the corresponding die size requirements for the PCA tile(s) to implement an AIM-like approach by the following steps.

- An earlier Pentium II processor also used the 0.35-micron process and had a total of 7.5 million transistors (about the same number as AIM). Its die size was 209 square mm. We used that to compute an estimate of the AIM die size (since the process and number of transistors were similar).
- We therefore assumed that the AIM chip would be slightly larger than 209 square mm.
- Using the projected 0.1 micron process which was assumed in the year 2000 Smart Memories paper [Mai], and the 3.5 squared factor of size reduction (from the 0.35 micron process), we calculated a 0.1 micron process die size for AIM of 20 square mm.
- The Smart Memories tile size was stated to be 2.5 x 2.5 mm, providing approximately 6 square mm of "real estate". Therefore, the AIM chip capability would need three such tiles. Therefore, we concluded that we could not utilize such a highly flexible crypto chip.
- A scaled-back approach to the crypto chip (without all the flexibility) should be much smaller, and we concluded that it could fit on one tile using the 0.1micron process. We do need to ensure that the crypto chip has an adequate random number generation capability included within it. (Among other things, it needs to generate a random number for an encryption key as part of the initialization process.)

While both MIT and Stanford used MIPS RISC processor cores, we also investigated the possible use of Intel Pentium and IBM/Motorola PowerPC cores to determine if they were a feasible alternative option. As the above analysis with the Pentium II showed, it would consume the space of three tiles to implement. More recent Pentium implementations have gotten even larger. Pentiums are CISC computers that would be expected to be physically larger. However, the PowerPC is a RISC-based chip, so we evaluated it as well. The corresponding numbers for at least one version of the PowerPC are as follows.

- For a 200 MHz PowerPC, using 0.35-micron process, the die size was 80 square mm.
- For 0.1 micron, the die size scales down to 6.5 square mm.
- This would fit on one tile (since the PowerPC considered here had about 2.5 million transistors compared to 7.5 million for the Pentium II).
- Some other PowerPC chips would be much larger, so this analysis does not apply to all PowerPC implementations.
- In comparison, the transistor count for the MIPS 4000 is 2.3 M while that for the MIPS 5000 is 3.0 M, and hence they each will fit on one tile.

Another potentially related technology is Digital Signal Processors (DSPs). There has been an established preference for special DSPs for applications such as radar processing. Hence, would the avionics PCA suite require DSP tiles? Work reported by MIT [Wentz] using Raw showed how it performs well in signal processing applications. We agree with the conclusion of the paper since the performance of general purpose processors is becoming increasingly competitive against specialized DSPs. We quote from "*Moore's Law and its Implications for Information Warfare*" [Kopp];

"*While a DSP is architecturally optimized for the application and will always perform better than a general purpose processor of similar complexity, running at the same clock speed, the question is whether the economics of production volumes will be able to sustain the DSP in the longer term. Cost pressures in hardware and development tools will increasingly favour the general purpose processor.*"

In addition to Smart Memories, Stanford University has also developed a technology called eXecute Only Memory (XOM), pronounced "zom" [Lie]. It is briefly described here since it might be used as a hardware base for the NSE. It is not an appropriate candidate for the AAP/EPE since it would be "over-kill" in some cases, and would not support the ability to assign XOM-protected software to a backup processor.

The XOM chip is useful when you have one or more of the following concerns:

1. You want to "lock" the application executable (and its license) to one specific computer system (i.e., you can't successfully copy it)
2. You want tamper-resistant application software
    a. You want to keep the program algorithms, etc. intellectual property secret
    b. You want to preclude anybody else from making changes to the application program
3. You want to create separate (isolated) data compartments without any data exchange between them
4. You want all data that leaves the chip to be encrypted (e.g., to main memory or a network)
5. You want support for a built in private key, private memory, and traps on cache misses
6. You don't want to have to trust the OS for access control (i.e., data separation)
7. You want protection against undetected data corruption in main memory (and register contents when swapped out and back in)
8. You also want backward compatibility with "normal" processing capabilities

All of these capabilities of XOM might make it a very good NSE, since all EPEs know the public key of the NSE, and the NSE needs to have its own private key (which XOM has in non-volatile storage for the private key). XOM provides integrity checks over all data accesses, which is a good feature for the NSE since any corruption of its security database could be very troublesome.

The encrypted executable software doesn't have any benefit beyond the "decrypt at load time" approach. There is no concern about somebody making a copy of the program or modifying it while it's on the air vehicle.

The separation of processes might be useful in some MLS aspects of the NSE.

The XOM approach would have merit in terms of the untrusted maintainer who has access through the maintenance panel. He/she could not read classified algorithms, etc. in the code. Also, the maintainer could not modify the code to insert any specific malicious routines. In addition, XOM provides an MD5 hash (MAC) authenticity and integrity check on the software as an on-going function.

XOM does not rely on the OS for security (e.g., partitioning). Different encryption keys keep the processes separated (except for the null key shared area which is another reason not to use it as the base for the EPE).

5.3 Elements of the MLS-PCA Model and Their Mapping to PCA

A key part of this task was to demonstrate the mapping of the formal security model to the PCA chip and its related resources. This involved several areas of investigation as listed below.

5.3.1 Analysis of the Specific Element Mappings

The following items list the elements of the model, and describe how each element is supported.

- The **AAP** is an untrusted process representing a given avionics application. AAPs have a variety of classification levels, but any specific AAP operates at a single security level. Each AAP is bound to one EPE when the AAP code is loaded into a specific 8-tile Region of Figure 5.1. The AAP and its associated EPE map to the crypto processor and memory tiles within the Region. The AAP can perform any morphing that it chooses within the bounds of the 6-tile portion of the Region in which it resides. For example, it can morph dynamically between parallel processing and pipelined processing.
- The **NSE** is a special trusted AAP, which may or may not be "housed" within a PCA chip. The trust issue is that it will be functionally MLS, since it deals with encryption keys that are classified to the level of the information that they protect.
- The **EPE** has trusted cryptographic functionality. With the exception of the EPE associated with the NSE, the EPEs operate at a single security level, namely that of the data being processed in the AAP to which they are bound. The EPE maps to the two encryption-related tiles of each "8-tile" Region

The combination of the EPE and AAP or NSE forms the basic building block for the MLS-PCA as shown earlier in Figure 5.1. The tiles within a region are interconnected via one or more (plain text) local bus(es), and the cipher text side of the crypto engine is connected to the global network of the PCA chip (and subsequently to the avionics interconnect network). The detail elements of the formal model are considered below.

- The **AAP/EPE binding** is at a single security level based on the avionics function that it has been assigned to perform. This binding maps to the bus interconnectivity within an 8-tile Region, i.e., the fact that the AAP processors and memory are connected to only the plain text side of the encryption tile. The EPE and AAP processors are "fate sharing" (as required by the model). This is accomplished by periodic software "keep alive" message exchanges.

- The **NSE/EPE binding** is at multiple security levels. It maps in the same way as the above AAP/EPE.
- A **"Trusted Connection"** exists between the NSE/EPE and each AAP/EPE. It maps to the network connectivity and the pair-wise unique common cryptographic key between the NSE/EPE and each AAP/EPE.
- The **Access Control Table** can be considered to be a matrix consisting of a list of all EPEs along both sides of the matrix. The cells of the matrix define the rights (if any) for "A" to connect to "B", and the policy enforcing key(s) that are to be utilized (XORed) in the generation of a session key. The contents of the access control table can be based on any policy, such as DAC. The initial content of the table is generated as part of mission planning and is securely loaded into the NSE at the start of a mission, but subsequent dynamic changes (by a trusted source) are supported by the model. It maps to memory within the NSE. Its implementation need not actually be an N x N matrix.
- **Simplex crypto connections** exist between EPEs. In most cases, there are two simplex connections between EPEs, one in each direction. However, the formal model allows one-way connections, such as to support write-up. They map to the network interconnectivity of the Region's global bus, and the placement of a unique common key at both end-point EPEs.
- The **crypto algorithm** per se is not specified in the model, but it is assumed to be adequate for the purpose, e.g., a Type 1 algorithm for DOD classified use. It maps to the hardware tile(s) upon which it operates. Key tables that are maintained in memory by the trusted EPE software also support it. It operates on the Region tile(s) that have hardware interconnections to both the cipher text bus and one or more of the plaintext busses.
- **Key management** consists of key generation, key distribution, rekey, revoke, and zeroize capabilities. There are special considerations for the support of coalition keying, including the distribution of the same keys to multiple parties (i.e., not just on a pair-wise basis. Key management maps to the NSE and EPE hardware, and the related key generation, key distribution, key usage, and key destruction/replacement.
- **The EPEs and the NSE collect audit data** when pre-defined security-relevant events occur. This data maps to the hardware memory locations where it is stored in each EPE and the NSE(s). At the end of the mission, this data would be written to a portable device where it would be collected and sent to a centralized site for analysis.
- **Secure boot/Initialization** consists of coming up in a secure state and trusted distribution of security-relevant code and data. The trust issue is the assurance that the downloaded code and data have a high level of integrity and the source has been authenticated. It maps to the security of the hardware boot mechanism and the availability of a secret (private) key in non-volatile memory that can be used for cryptographic signature verification.

The PCA hardware is usually a set of resources over which the compiler has complete control. With MLS-PCA, the EPE crypto and processor/memory are exceptions to that rule. Therefore, the SAPI and/or SAAL "view" of the PCA tiles would need to be restricted with regard to the EPE tiles. There are a variety of hardware mapping issues that are more broadly based than the above direct mapping between the elements of the formal model and the hardware. These are discussed in the remainder of this section.

Tile and bus grouping and layouts: There should be a number of different types of tiles, including CPU tiles, memory tiles, and crypto tiles. The crypto and EPE tiles are allocated to the two end tiles in an 8-tile Region. They are at the edge of the chip, and the cipher text bus is connected to the global network. The remaining Region tiles consist of processor and 8 MB DRAM memory tiles for the AAP or NSE. This Region needs to have at least one bus to interconnect the AAP (or NSE) processor and memory tiles, and also interconnects them to the plaintext side of the crypto tile. A separate bus connects the cipher text side of the crypto tile to the global network.

5.3.2 Required constraints to achieve MLS

A fundamental constraint of the MLS-PCA architecture is that that there must not be any operational data flow that bypasses the crypto tile(s). This does not preclude some morphed bypass that may be needed to initialize the system. There must also be a mechanism that securely passes the plain text routing header on encrypted packets.

A corollary constraint is that there cannot be any interconnect between 8-tile Regions except on the cipher text side. This is one of the most critical constraints on the morphing. The "wires" that would otherwise exist at these points must be securely severed, which we accomplished by etching away any such potential interconnectivity.

The shared busses must only convey cipher text and possibly unclassified information. This constraint is the one that is most "against the grain" of the PCA architectures.

5.3.3 Specific mapping to Smart Memories and Raw PCA chips

The following points summarize the specific issues that we found in attempting to map the formal model to the Raw and Smart Memories architectures.

- Smart Memories supports four intra-quad busses. We need at least two; one for the plain text and the other for the cipher text and connection to the global network. As discussed earlier, we need for these busses to span an adjacent pair of quads. Raw does not provide any dedicated (non-multiplexed) busses within a Region (or any where else).
- Smart Memories morphs the configuration (including the Region boundaries) by a set of registers that can be controlled by the support processor, or by a privileged on-chip processor. Raw has a "static network" which is pre-programmed by the compiler for register-level transfers anywhere on the chip. Hence, this program

would have to be trusted to maintain the Region boundaries. As indicated previously, “the static routers collectively reconfigure the entire communication pattern of the network on a cycle-by-cycle basis” [Taylor 03].

- Neither chip provides non-volatile on-chip memory (e.g., for the public keys).
- Smart Memories provides the option to use some tiles for DRAM. Raw only supports off-chip DRAM. (This is partly due to the fact that IBM, who made their chip, did not have embedded DRAM in their Application Specific Integrated Circuit (ASIC) library.)
- Neither currently supports the crypto tile(s).

5.3.4 Assumptions

We found it necessary to make the following assumptions about the future PCA chip capabilities to support the MLS-PCA Regions.

- A DRAM memory tile can contain a total of 8 MB of instructions and/or data.
- The 8-tile Region-defining configuration data can be “locked down” securely, but still allow the individual AAPs to morph their computing resources.
- The EPE processor and 4 MB of instruction/data memory will fit on one tile.
- We will be able to find some way to support at least a limited amount of non-volatile memory, e.g., by including some anti-fuse capabilities on the chip.
- Intra-Region dedicated busses will be available across the entire 8-tile Region.
- A crypto tile (including an adequate random number generator) will be available.
- Moore’s law will continue to apply.

5.4 A Summary of Current PCA Limitations for MLS Avionics

The following is a summary list of the changes that are needed in order to support MLS-PCA.

- Shared memory, memory management, and process scheduling disabled between Regions
- Substantial increase in the amount of on-chip DRAM
- Partitioning access to off-chip DRAM at differing security levels (if the on-chip amount is inadequate)
- Provision of non-volatile on-chip memory to support initialization
- Support for the crypto tile(s)
- High assurance “nail down” of the Region boundaries (avoiding any inter-Region communications other than via the crypto modules)
- More access to dedicated bus/network paths
- Region-specific morphing
- Concrete metadata restrictions prevent violating intra Region constraints, e.g., all external bus communications between Regions pass through the Region’s EPE first on input to, and last on output from an AAP

The above list of MLS constraints in applying the current PCA architectures to DOD avionics is not surprising given that we are applying PCA in a completely different way than was originally intended. These are fixable matters. The "really good news" is that there are many positive points as well.

First of all, the MLS-PCA approach once and for all solves the critical "MLS risk issue" that has plagued modern piloted air vehicle development, and will continue to be of concern with future uninhabited air vehicles (UAVs) and uninhabited combat air vehicles (UCAVs). Another benefit will be the replacement of the bulky, power-hungry, on-board super computers that are in use today with a few PCA chips in future systems. Size, weight, energy consumption, and such will become even more important than they are today. One can expect that future avionic systems will continue to grow in both lines of code and the need for processing power for signal processing, information fusion, artificial intelligence, etc. This will be a challenge for future PCA-like developments, but PCA appears to be the best available solution to date.

6 CONCLUSION

With this final report, Northrop Grumman Corporation has successfully completed all tasks for which it was contracted. More importantly, it has achieved significant results in developing a MLS architecture, MLS-PCA, which is flexible enough to address the pressing security needs of DOD in its march forward to its network-centric vision of the 2020 battle space. MLS-PCA is also scalable to very large distributed avionics applications that may execute on networks of 400,000 processors within future aircraft, UAVs, ships and ground systems.

MLS-PCA was successfully designed, modeled, formally (i.e., mathematically) specified and validated by the MIT Alloy specification language and constraints checker. The architecture is simple, which will enable high assurance application development in the future at a fraction of current Certification and Accreditation efforts. Furthermore, we have presented a number of ways that the PCA hardware and morphware need to be constrained in order to achieve a high level of assurance. Given these constraints, MLS-PCA has been shown to map to the new DARPA PCA chips, giving DARPA a path to security and performance with the new chip technology.

Much has been accomplished, but much work remains to build MLS-PCA for the new DARPA PCA chips. Toward that end, Northrop Grumman Corporation has initiated a prototype implementation demonstration with its 2003 IR&D funds. The MLS-PCA demo will execute a distributed targeting algorithm set of AAPs operating on a Grid Computing network in Northrop Grumman Corporation R&D lab. Grid Computing is the only way to simulate thousands of processors decades before the actual chips exist. The demo will simulate MLS data traffic over the grid protected by MLS-PCA. The prototype will be a proof-of-concept demo, and a vehicle to gather performance data. DARPA can capitalize on this in subsequent follow-on developments. The experience gained by implementing the prototype will be used to guide the next round of implementation of MLS-PCA on real PCA chips.

7 ACKNOWLEDGMENTS

The Northrop Grumman Corporate team that performed this research consisted primarily of Dr. Gerald (Jerry) Cole, Brant Hashii, and Clark Weissman, with support from Matthew Allison, Jeffrey Barnett, and Barry Gold of the Avionics & Software Systems Information Assurance group.

Special thanks are due to the following persons from other PCA teams, who provided valuable feedback and support in performing this research: Professor Mark Horowitz, Ron Ho, and David Lie of the Stanford Smart Memories project, as well as that of Michael Taylor of the MIT Raw project.

8 REFERENCES

[Agents] Intelligent Software Agents Lab. Available at http://www.cs.cmu.edu/~softagents.

[AIM] AIM 1998. Programmable cryptography emerges from advanced chip, Signal Magazine. August

[Agrawal] AGRAWAL, ANANT 1999. Raw computation. Scientific American, August

[Anderson] ANDERSON, J. P. 1972. Computer Security Technology Planning Study. In ESD-TR-73-51.

[BLACKER] WEISSMAN, C. 1988. BLACKER: Security for the DDN, Examples of A1 Security Engineering. Presented at 1988 IEEE Symposium on Security and Privacy. In Proceedings 1992 Conference IEEE Symposium on Security and Privacy, pp 286. Available at http://www.computer.org/proceedings/sp/2825/28250286abs.htm

[Bell] BELL, D. E., AND LA PADULA, L. 1975. Secure Computer Systems: Unified Exposition and Multics Interpretation. In Technical Report ESD-TR-75-306, ESD/AFSC, Hanscom AFB, Bedford, MA. Available at http://csrc.nist.gov/publications/history/.

[Campbell] CAMPBELL, DANIEL et al, 2002. Section 3: PCA architectures. At: www.morphware.org/facilitating%20Middleware%20DC%20final%20012203.pdf

[CC99] Common Criteria for Information Technology Security Evaluation. ISO/IEC 15408, Version 2.1, CCIMB-99-031, August 1999. Available at http://www.radium.ncsc.mil/tpep/library/ccitse/ccitse.html.

[CSC003] CSC-STD-003-85. Computer Security Requirements – Guidance for Applying the DOD TCSEC in Specific Environments. June 1985. Available at http://www.fas.org/irp/nsa/rainbow.htm.

[CSC004] CSC-STD-004-85. Technical Rationale Behind CSC-STD-003-85: Computer Security Requirements. June 1985. Available at http://www.fas.org/irp/nsa/rainbow.htm.

[DCID6/3] Protecting Sensitive Compartmented Information within Information Systems. Director of Central Intelligence Directive 6/3. June 1999**.** Available at http://www.fas.org/irp/offdocs/DCID_6-3_20Policy.htm.

[Diffie 76] DIFFIE, W. AND HELLMAN, M. E. 1976. Lattice-based access control models. IEEE Transactions on Information Theory IT-22, 6 (Nov.), 644–654.

[Diffie 92] DIFFIE, W., VAN OORSCHOT, P. C., AND WIENER, M. J. 1992. Authentication and authenticated key exchanges. Designs, Codes and Cryptography 2, 2, 107–125.

[DITSCAP] DOD Information Technology Security Certification and Accreditation Process. DITSCAP. DOD 5200.40, December 1997. Available at http://www.dss.mil/infoas/index.htm

[DODD 8500] Information Assurance. DOD Directive 8500.1, October 2002. Available at http://www.dtic.mil/whs/directives

[DODI 8500] Information Assurance (IA) Implementation. DOD Instruction 8500.2, February 2003. Available at http://www.dtic.mil/whs/directives

[GridC] Grid Computing Center. Available at http://www.gridcomputing.com/

[Griff] GRIFFIOEN, D. AND HUISMAN, M. 1998. A comparison of PVS and Isabelle/HOL. In Theorem Proving in Higher Order Logics: 11th International Conference, TPHOLs '98, J. Grundy and M. Newey, Eds. Lecture Notes in Computer Science, vol. 1479. Springer-Verlag, Canberra, Australia, 123–142.

[Hashii01] HASHII, B. 2001. Formal Specification Languages and Theorem Provers. Northrop Grumman, December 2001.

[Hashii03a] HASHII, B. 2003. Using Alloy to Formally Specify MLS-PCA Trusted Security Architecture. Northrop Grumman, July 2003.

[Hashii03b] HASHII, B. AND ALLISON, M. 2003. Multi Level Security Polymorphous Computing Architecture Formal Alloy Model, Version 2.4. Northrop Grumman, May 2003.

[Ho] HO, RON, et al. 2001. The future of wires. Proceedings of the IEEE, April 2001

[Horowitz] HOROWITZ, MARK, et al 2000. Stanford – PCA morphware virtual machine plan. Available at: graphics.stanford.edu/sss/VMDesignPlan.pdf

[Jackson96] JACKSON, D., AND WING, J. M. 1996. Lightweight Formal Methods. In IEEE Computer, pp 21-22, April 1996.

[Jackson01a] JACKSON, D. 2001. Micromodels of Software: Modelling & Analysis with Alloy. MIT Lab for Computer Science, November 2001. Available at http://sdg.lcs.mit.edu/alloy/book.pdf.

[Jackson01b] JACKSON, D., SHLYAKHTER, I., AND SRIDHARAN, M. 2001. A micromodularity mechanism. In Proceedings of the ACM SIG-SOFT Conference on the Foundations of Software Engineering / European Software Engineering Conference. Vienna, Austria, 62–73.

[JV2020] Joint Vision 2020. JCS, J5, June 2000. Available at http://www.dtic.mil/jointvision/jvpub2.htm.

[Karger] KARGER, P. A. 1989. New methods for immediate revocation. In 1989 IEEE Symposium on Security and Privacy. IEEE Computer Society, Oakland, CA, USA. Available at http://www.multicians.org/biblio.html.

[Kopp] KOPP, CARLO 2002. Moore's law and its implications for information warfare.Invited Paper, 3rd International AOC EW Conference, January 6

[Kouts] KOUTSOFIOS, E. AND NORTH, S. 2002. Drawing graphs with dot. AT&T Labs-Research. Available at http://www.research.att.com/sw/tools/graphviz/refs.html.

[Lie] LIE, DAVID, et al 1999. Hardware support for copy and tamper resistant software. Stanford University. Available at http://wwwvlsi.stanford.edu/~lie/Talks/xom-asplos.pdf

[Locasso] LOCASSO, R., SCHEID, J., SCHORRE, V., AND EGGERT, P. 1980. The Ina Jo specification language reference manual. Tech. Rep. TM-(L)-6021/001/00, System Development Corporation. June.

[Mai] MAI, KEN, et al, 2000. Smart memories: A modular reconfigurable architecture", Stanford University

[NIST] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. 2003. Special Publication 800-57 recommendation for key management part 1: General guideline. Draft, Available at http://csrc.nist.gov/CryptoToolkit/tkkeymgmt.html.

[NIAP] National Information Assurance Partnership, NIAP. NIST, 1997. Available at http://niap.nist.gov/

[NISPOM] National Industrial Security Program Operating Manual, NISPOM. DOD 5220.22-M, December 1993. Available at http://www.dss.mil/infoas/index.htm

[NSTISSC] National Security Telecommunications and Information Systems Security Committee, NSTISSC, Policy #11. July 2002. Available at http://niap.nist.gov/cc-scheme/nstissp_11.pdf

[Moore] Definition. Available at http://www.webopedia.com/TERM/M/Moores_Law.html

[PCA] Polymorphic Computing Architecture Mission. Available at http://www.darpa.mil/ipto/research/pca/

[PKPP] Partitioning Kernel Protection Profile, Preliminary Draft V0.3. NSA C12, October 2002.

[Rainbow] Rainbow Series of books on evaluating Trusted Computer Systems according to National Security Agency (NSA) expounding on the Orange Book [TCSEC]. Available at http://www.fas.org/irp/nsa/rainbow.htm

[Richards] RICHARDS, MARK, et al, 2003. The morphware stable interface: A software framework for polymorphous computing architectures. Available at www.morphware.org/GOMAC_03_paper_Richards.pdf

[Saaltink] SAALTINK, M. 1997. The Z/EVES user's guide. Tech. Rep. TR-97-5493-08, ORA Canada, 267 Richmond Road, Suite 100, Ottawa Ontario K1Z 6X3, Canada. Sept. Available at http://www.ora.on.ca/z-eves/documentation.html.

[Sandhu] SANDHU, R. S. 1993. Lattice-based access control models. IEEE Computer 26, 11, 9–19. Available at http://citeseer.nj.nec.com/article/sandhu93latticebased.html.

[SoC NV] SoC NV, 2002. Non-volatile embedded memory on a standard logic process, ECN Magazine, April

[Spivey] Spivey, J. M. 1992. The Z Notation: A Reference Manual, 2nd ed. Prentice Hall International Series in Computer Science.

[Taylor 02] Taylor, Michael 2002. The Raw prototype design document V4.11", MIT, December

[Taylor 03] Taylor, Michael, et al, 2003. A 16-issue multiple-program-counter microprocessor. IEEE International Solid-State Circuits Conference, February/Session 9.

[TCSEC] Department of Defense Trusted Computer System Evaluation Criteria (TCSEC). DOD 5200.28-STD, December 1985. Available at http://www.fas.org/irp/nsa/rainbow.htm TCPA

[Tirpak] Tirpak, J. A. 2002. The F-22 on the line. Air Force Magazine 85, 09. Available at http://www.afa.org/magazine/Sept2002/0902raptor.html.

[TCPA] Trusted Computing Platform Alliance. 2002. Trusted Computing Platform Alliance (TCPA) Main Specification Version 1.1b. Available at http://www.trustedcomputing.org/tcpaasp4/index.asp.

[Wentz] Wentzlaff, David, et al, 2001. The Raw architecture: signal processing on a scalable composable computation fabric. Presented at the High Performance Embedded Computing Workshop

[Xilinix] Xilinx 2003 Virtex series configuration architecture user guide. Xilinx Corporation, March 24

[Young] Young, W. D. 1997. Comparing verification systems: Interactive consistency in ACL. An earlier version appears in Proceedings of the Eleventh Annual Conference on Computer Assurance, pages 35-45, 1996. Also available at http://www.cs.utexas.edu/users/moore/publications/acl2-papers.html.

9 ACRONYMS

AA	Alloy Analyzer
AAP	Avionics Application Process
ACL2	A Computational Logic for Applicative Common Lisp
AES	Advanced Encryption Standard

AIM	Advanced INFOSEC Machine
ASCII	American Standard Code for Information Interchange
ASIC	Application Specific Integrated Circuit
BLP	Bell-LaPadula
C&A	Certification and Accreditation
CC	Common Criteria
CISC	Complex Instruction Set Computer
COTS	Commercial Off the Shelf
CPU	Central Processing Unit
DAC	Discretionary Access Control
DARPA	Defense Advanced Research Projects Agency
DDN	Defense Data Network
DES	Data Encryption Standard
DOD	Department of Defense
DRAM	Dynamic Random Access Memory
DSP	Digital Signal Processor
EPE	Encryption Processing Element
FPGA	Field Programmable Gate Array
GPS	Global Positioning System
I&A	Identification and Authentication
IBM	International Business Machines
IPC	Inter-Process Communications
IR	InfraRed
IRAD	Independent Research and Development
JIAWG	Joint Integrated Avionics Working Group
JV2020	Joint Vision 2020
LAN	Local Area Network
MAC	Message Authentication (Integrity) Check, and also Mandatory Access Control
MD5	Message Digest 5 (a hash algorithm)
MIPS	MIPS Technologies Company
MIT	Massachusetts Institute of Technology
MLS	Multi-Level Security
MPC	Mission Planning Center
NSE	Network Security Element
NSTISSC	National Security Telecommunications and Information Systems Security Committee
OS	Operating System
PCA	Polymorphous Computing Architectures
pid	process identifier
PK	Partitioning Kernel
PKI	Public Key Infrastructure
PKPP	Partitioning Kernel Protection Profile
PMD	Portable Memory Device
PVS	Prototype Verification System
R&D	Research and Development

RISC	Reduced Instruction Set Computer
SAAL	Stable Architecture Abstraction Layer
SAPI	Stable Application Programming Interface
SLOC	Source Lines of Code
SoC	System on a Chip
SRAM	Static Random Access Memory
SSO	System Security Officer
TC	Trusted Connection
TCB	Trusted Computing Base
TCPA	Trusted Computing Platform Alliance
TCP/IP	Transmission Control Protocol / Internet Protocol
TCSEC	Trusted Computer System Evaluation Criteria
TS-SAR	Top Secret – Special Access Required
UACV	Uninhabited Combat Air Vehicle
UAV	Uninhabited Air Vehicle
uid	user identifier
UML	Unified Modeling Language
VDM	Vienna Development Method
VPN	Virtual Private Network
XOM	eXecute Only Memory